Charity General Hospital Database

CS342 Winter 2016

Alex Rinaldi & JoAnn Tuazon

Table of Contents

1. Fact Finding, Information Gathering, and Conceptual Database Design.

To build a database for Charity General Hospital, we need to design a concept for the structure of the database based on the needs and requirements of the organization. In this section, we will create a conceptual database design for Charity General Hospital. We will describe methods for gathering data, and then present our conceptual database design in detail.

1.1 Fact-Finding Techniques and Information Gathering

In order to build a conceptual database for an organization, designers must understand the organization. This section gives a brief description of the organization, an explanation of the research process for designing a conceptual database and how we applied it to our organization, a description of the specific problem our database is trying to solve for our organization, and itemized descriptions of the entities and relationships that form our design.

1.1.1 Introduction to Organization

Charity General Hospital is a fictitious general hospital designed to combine similar aspects from two general hospitals in Bakersfield – *Mercy Hospital* and *San Joaquin Community Hospital*. Charity offers medical care and temporary residence for patients with a large variety of health needs including post-surgery care, heart health, and stroke care. Groups of nurses at *Charity* continuously visit the rooms of all patients to assess health and provide services based on each individual patient's needs, such as administering medication or cleaning wounds. Nurses are required to keep track of all of the different services required by patients during their shift, as well as document each service offered.

1.1.2 Description of Fact-Finding Techniques

Forming a conceptual database requires detailed information on the kind of data that will be stored and how it will be used by members of the organization – this process is called requirements collection and analysis. To gather information, designers should interview and survey potential users of the database to find out the underlying structure of the organization, as well as what information they would like to regularly access and manipulate. Researching the organization's documents and existing database systems, as well as observing regular processes that occur in the organization is also useful.
To gather information about our organization, we interviewed nurses from both *Mercy Southwest* and *San Joaquin Community* hospitals and studied their paper documentation process. This helped us form data requirements – requirements detailing what data needs to be stored and its structure. At general hospitals, historical data is kept about every order issued by doctors for patients, as well as every action carried out by nurses for patients.

Interviewing the nurses helped us also form functional requirements – what kind of operations is performed on the data. Doctors add historical records about every prescription and order they make. Nurses access doctors' orders to determine what actions they need to take with patients. The nurses whom we interviewed also commented that they must document every single action they perform with patients for legal reasons.

1.1.3 Scope of the Conceptual Database

When designing a conceptual database, it is important to have a clear idea of the part of the organization that the database represents. Sometimes, this is called the "miniworld."
General hospitals record all information about the services patients require (prescriptions and special needs ordered by doctors) and all actions performed by nurses through a paper documentation process. While this largely can't be changed for legal reasons, our goal with the database is to create a more efficient system for storing historical records and accessing all the doctor-ordered patient needs at once. Our database is meant to generalize the recording process across all units, so special equipment required for each unit (like the ICU) is not part of the conceptual design. Also, to ensure that only information relevant to nurses working on site with patients is stored, our database excludes information about "out patients" (patients who are not in residence at the hospital). Also, operating rooms are excluded from the scope of the database.

1.1.4 Entity and Relationship Sets Description

Once we gather all of the information about how the hospital data is structured, we can represent the data as entity sets and relationship sets using the Entity-Relationship (ER) model. Section 1.2 will explain the purpose of entity sets and relationship sets in greater detail, but this section will provide a basic description of each of the "objects" that make up a general hospital (entity type definitions), as well as how they are related to each other (relationship type definitions).

Entity type definitions

Employee: EmployeeID, SSN, Name, Address, Phone Number
An **employee** works for the hospital, and has basic identifying and contract information.

Doctor: License number
A **doctor** is a specialization of an employee. They issue orders about how nurses should treat patients.

Nurse: License number
A **nurse** is a specialization of an employee. They perform the actions ordered by doctors and assess how patients respond.

Prescription: Prescription ID, Dosage, Frequency, Start date, End date
A **prescription** is contains a medication ordered by a doctor for a single patient, including administration instructions.

Medication: Medication ID, Medication Name, Medication Purpose, Dosage, Frequency
A single **medication** belongs to a Prescription.

Room: Room ID, Room Number
A **room** houses a patient in the hospital and has a unique number.

Unit: Unit ID, Unit Name, Unit Abbreviation
A **unit** has a specific location in a general hospital and contains many rooms.

Assessment: Assessment ID, Blood pressure, Respiration rate, Heart rate
An **assessment** is taken by a nurse for a single patient, and describes basic information about their current condition.

Special Need: Special Need ID, Frequency, Start date, End date
A **special need** is an activity that needs to be performed by a nurse on a single patent a certain number of times over a time period.

Activity: Activity ID, Activity Name, Description
A single **activity** belongs to a special need order. It describes the content of the special need order.

Patient: Patient ID, SSN, Name, Address, Phone number, Birth date, Gender, Insurance number, Admitted date, Discharged date, Language spoken
A **patient** is admitted to the hospital, and contains basic identifying and contact information, as well as language spoken in case a translator is needed.

Relationship type definitions

Doctor **prescribes** Prescription; *Cardinality*: 1..N; *Participation*: Partial, Total (respectively)

Prescription **prescribed for** Patient; *Cardinality*: N..1; *Participation*: Total, Partial

Prescription **hasMedication** Medication; *Cardinality*: N..1; *Participation*: Total, Partial

Nurse **administers** Prescription; *Cardinality*: M..N; *Participation:* Partial, Partial

Nurse **takes** Assessment; *Cardinality*: 1..N; *Participation*: Partial, Total

Patient **undergoes** Assessment; *Cardinality*: 1..N; *Participation*: Partial, Total

Doctor **orders** Special Need; *Cardinality*: 1..N; *Participation*: Partial, Total

Nurse **performs** Special Need; *Cardinality*: M..N; *Participation*: Partial, Partial

Patient **receives** Special Need; *Cardinality*: 1..N; *Participation*: Partial, Total

SpecialNeed **has Activity** Activity; *Cardinality*: N..1; *Participation*: Total, Partial

Nurse **assigned to** Patient; *Cardinality*: M..N; *Participation*: Partial, Partial

Nurse **works in** Unit; *Cardinality*: M..N; *Participation*: Total, Total

Patient **placed** Room; *Cardinality*: M..N; *Participation*: Total, Partial

Room **belongs to** Unit; *Cardinality*: N..1; *Participation*: Total, Total

1.2 Conceptual Database Design

Before designing a physical system to store the data we gathered in the previous section, we have to understand how this data will be stored. One way to represent the structure of data is using the Entity-Relationship (ER) model. Using this method, we try to represent the structure of the hospital data using two concepts: *entity sets*, which define real-world "objects" (like people, places, and things); and *relationship sets,* which define how entities of two or more entity types are related.

We create *entity types* to unite entity sets that represent the same object - like a "Nurse" - by naming and defining each of the attributes that "Nurse" contains. We also create *relationship types* to define how entity types are related to each other, as well as any other attributes describing how they are related.

In this section, we will define each entity type in detail, including each attribute and its domain. Then we will define each relationship: the entity types each relates, the cardinality, and participation. Finally, we will present a diagram that visually represents the conceptual design.

1.2.1 Entity Set Description

Entities describe real world-objects, and are defined by their name and the attributes they contain. In our model, the most important entities are doctors, nurses, and patients. Also important are the orders given by doctors (prescriptions or special needs) and individual assessments of patients.
We now list each entity type and its name, the attributes it defines, the domain constraints of each attribute, fields to be indexed for fast access, and keys, or fields that uniquely identify individual entities in the set.

Entity Name: Employee

Description: An employee is paid by the hospital to work on-site during shifts. Since the scope of our database includes only positions related to patient care, employees can either be doctors or nurses (a total, disjoint specialization). The database stores basic identifying and contact information about Employees, as well as the periods of their employment. Employees will be infrequently inserted and updated by HR workers at the hospital.

Candidate Keys: Employee ID, SSN
Primary Key: Employee ID
Entity Type: Strong
Fields to be Indexed: Employee ID, Name, SSN

Attributes:

Attribute Name	Employee ID	SSN	Name
Description	Number assigned by the hospital to each Employee that uniquely identifies them. Can be implemented as an auto-increment integer.	Social Security Number; can be used to uniquely identify Employees.	Name of Employee (First, Middle, Last)
Domain / Type	Integer	Integer	String, String, String
Value / Range	0 - MaxID	000000000 - 999999999 (any 9-digit Integer)	Any, Any, Any
Default Value	MaxID + 1	None	None
Null Value Allowed	No	No	No
Unique	Yes	Yes	No
Single or Multi-value	Single	Single	Single
Simple or Composite	Simple	Simple	Composite

Employee Attributes Continued:

Attribute Name	Address	Phone Number	Start Date	End Date
Description	Street Address, City, State, Zip Code	Primary contact number	Starting date of employment period(s) (hence multi-value)	Ending date employment period(s).
Domain / Type	String, String, String, Integer	Integer	Date	Date
Value / Range	Any, Any, Any, 00000-99999	0000000000 - 9999999999	Any	Any
Default Value	None	None	None	None
Null Value Allowed	No	No	No	Yes
Unique	No	No	No	No
Single or Multi-value	Single	Single	Multi-value	Multi-value
Simple or Composite	Composite	Simple	Simple	Simple

Entity Name: Doctor

Description: Doctors work on site at the hospital, but they are less directly involved with patients than nurses. They decide which actions Nurses need to perform on each Patient (including when, for how long, and how often) through two types of orders: Prescriptions (for medication) and Special Needs (for required activities like assisted showers, bed turning for bedridden patients, or wound cleaning for patients with wounds).

Candidate Keys: License Number
Primary Key: License Number
Strong / Weak Entity: Strong
Fields to be indexed: License Number

Attributes:

Name	License Number
Description	All California doctors are given an "MD" (Medical Doctor) license number that uniquely identifies them from all other doctors in California. The number has two parts: the letters "MD" and a unique 6-digit number
Domain / Type	String, Integer
Value / Range	All 6-digit Integers preceded by the string "MD"
Default Value	None
Null Value Allowed	No
Unique	Yes
Single or Multi-value	Single
Simple or Composite	Composite

Entity Name: Nurse

Description: Nurses work on sites at the hospital. During one shift, each nurse is assigned to a single room and takes care of the patient who inhabits it (in our model, a nurse is directly assigned to a patient for simplicity). A nurse's main responsibility is to carry out the activities and administer the medication for their assigned patient as ordered by a doctor. Nurses also periodically record vital signs information about their patient as part of a Nursing Assessment (at Mercy, this happens every four hours). Nurses must record every single action they take with their patient during a shift.

Candidate Keys: License Number
Primary Key: License Number
Strong / Weak Entity: Strong
Fields to be indexed: License Number

Attributes:

Name	License Number
Description	All California nurses are given an "RN" (Registered Nurse) license number that uniquely identifies them from all other nurses in California. The number has two parts: the letters "RN" and a unique 6-digit number
Domain / Type	String, Integer
Value / Range	All 6-digit Integers preceded by the string "RN"
Default Value	None
Null Value Allowed	No
Unique	Yes
Single or Multi-value	Single
Simple or Composite	Composite

Entity Name: Prescription

Description: Prescription represents a single drug assigned to a single patient. It is related to a medication through the "has medication" relationship. Doctors prescribe medication to patients based on their needs, and Nurses make sure that patients receive the medication prescribed by Doctors. Nurses administer each medication multiple times throughout the patient's stay in the hospital based on the "Frequency" and "Start/End Date" fields. Prescriptions are inserted as the doctor orders them and are never updated or deleted (only replaced by new prescriptions).

Candidate Keys: Prescription ID
Primary Key: Prescription ID
Strong / Weak Entity: Strong
Fields to be indexed: Prescription ID

Attributes:

Attribute Name	Prescription ID	Dosage	Frequency
Description	An auto-increment unique identifier for prescriptions.	Describes the amount that should be administered in a dose.	How often the dose should be administered.
Domain / Type	Integer	String	String
Value / Range	0 - MaxID	Any	Any
Default Value	MaxID + 1	None	None
Null Value Allowed	No	No	No
Unique	Yes	No	No
Single or Multi-value	Single	Single	Single
Simple or Composite	Simple	Simple	Simple

Prescription Continued:

Attribute Name	Start Date / Time	End Date / Time
Description	The date the medication should first be administered.	The date when the medication should no longer be administered.
Domain / Type	Date / Time	Date / Time
Value / Range	Any	Any
Default Value	None	None
Null Value Allowed	No	No
Unique	No	No
Single or Multi-value	Single	Single
Simple or Composite	Simple	Simple

Entity Name: Medication

Description: A single medication is tied to each prescription when a Doctor prescribes it. The same medication can appear in multiple prescriptions. Each medication has a descriptive name that indicates its contents, and a summary of its purpose (for example, “painkiller”).

Candidate Keys: Medication ID, Medication Name
Primary Key: Medication ID
Strong / Weak Entity: Strong
Fields to be indexed: Medication ID, Medication Name, Medication Purpose

Attributes:

Attribute Name	Medication ID	Medication Name	Medication Purpose
Description	An auto-increment unique identifier for Medication.	Describes the chemical contents of the medication.	Brief description of the function of the medication.
Domain / Type	Integer	String	String
Value / Range	0 - MaxID	Any	Any
Default Value	MaxID + 1	None	None
Null Value Allowed	No	No	No
Unique	Yes	No	No
Single or Multi-value	Single	Single	Single
Simple or Composite	Simple	Simple	Simple

Entity Name: Assessment

Description: A nurse conducts a basic assessment of a patient's health periodically. In the Medical/Surgical unit, this happens once every four hours. The information included in the assessment is generally the same for all units - blood pressure, respiration rate, and heart rate. Assessments are never updated, but are inserted frequently throughout the day as they are performed.

Candidate Keys: Assessment ID
Primary Key: Assessment ID
Strong / Weak Entity: Strong
Fields to be indexed: Assessment ID

Attributes:

Attribute Name	Assessment ID	Blood Pressure	Respiration Rate	Heart Rate
Description	An auto-increment integer to uniquely identify assessments	Systolic and diastolic blood pressure readings in millimeters of mercury	Rate at which the patient is breathing	Patient's current heart rate in bpm
Domain / Type	Integer	Integer, Integer	Integer	Integer
Value / Range	0 – MaxID	0 – 300, 0-300	0 – 300	0 – 300
Default Value	MaxID + 1	None	None	None
Null Value Allowed	No	No	No	No
Unique	Yes	No	No	No
Single or Multi-value	Single	Single	Single	Single
Simple or Composite	Simple	Composite	Simple	Simple

Entity Name: Special Need

Description: A special need describes an activity for which a specific patient requires assistance from a nurse. They include the name and description of the activity (e.g. bed turn, wound cleaning, assisted shower) along with how often the activity should be performed. Each special needs activity is specific to a single patient. A nurse usually performs Special needs several times a day. The doctor's orders determine which special needs a patient requires, including how often the special need should be addressed. Special needs are inserted, never updated or deleted.

Candidate Keys: Special Need ID
Primary Key: Special Need ID
Strong / Weak Entity: Strong
Fields to be indexed: Special Need ID

Attributes:

Attribute Name	Special Need ID	Frequency	Start Date / Time	End Date / Time
Description	An auto-increment unique identifier for Special Needs	Describes how often a patient requires an activity ("once a day," "once a week")	The date the special need activity should first be performed.	The date when the special need activity should no longer be performed.
Domain / Type	Integer	String	Date / Time	Date / Time
Value / Range	0 – MaxID	Any	Any	Any
Default Value	MaxID + 1	Null	None	None
Null Value Allowed	No	Yes	No	No
Unique	Yes	No	No	No
Single or Multi-value	Single	Single	Single	Single
Simple or Composite	Simple	Simple	Simple	Simple

Entity Name: Activity

Description: An activity is specified by a Special Need order from a doctor. It contains a name that describes the activity, along with a more detailed activity. The same activity can appear in multiple special need orders.

Candidate Keys: Activity ID, Activity Name
Primary Key: Activity ID
Strong / Weak Entity: Strong
Fields to be indexed: Activity ID, Activity Name

Attributes:

Attribute Name	Activity ID	Activity Name	Activity Description
Description	An auto-increment unique identifier for Activity	Wound care, bed turning, catheter car, etc.	Specific information about the activity the nurse must know for certain patients.
Domain / Type	Integer	String	String
Value / Range	0 – MaxID	Any	Any
Default Value	MaxID + 1	None	Null
Null Value Allowed	No	No	Yes
Unique	Yes	No	No
Single or Multi-value	Single	Single	Single
Simple or Composite	Simple	Simple	Simple

Entity Name: Room

Description: A room temporarily houses a single patient. For the most part, rooms in general hospitals are all designed to provide all necessary services to all patients in a unit. All rooms have a number and are located in a unit. Nurses repeatedly visit a patient in a room to administer medication and fulfill special needs – it is important to know a patient's room so they can be located. Unless the hospital building expands, rooms generally do not have to be updated or deleted.

Candidate Keys: Room ID, Room Number
Primary Key: Room ID
Strong / Weak Entity: Strong
Fields to be indexed: Room ID, Room Number

Attributes:

Attribute Name	Room ID	Room Number
Description	Auto-increment integer that uniquely identifies a room.	Uniquely identifies a room. Usually contains information about the unit it belongs to and the floor it is on.
Domain / Type	Integer	Integer
Value / Range	0 – MaxID	0 – 999
Default Value	MaxID + 1	None
Null Value Allowed	No	No
Unique	Yes	Yes
Single or Multi-value	Single	Single
Simple or Composite	Simple	Simple

Entity Name: Unit

Description: A unit describes a set of rooms that offer the same level of service to patients. Units have abbreviated names so they can easily be identified. Examples are the Intensive Care Unit (ICU), which offers health care to critically ill patients, or the Medical/Surgical Unit, which offers less intense care to patients who are recovering from surgery. The condition of the patient determines which unit they are sent to. (In our model, the same, generalized framework is used to describe services offered to patients belonging to all units). Units generally have a fixed location in the hospital, such as a floor or an area of a floor (in this way, units can be used to locate rooms). New units will need to be inserted if a hospital expands, but by no means regularly.

Candidate Keys: Unit ID, Unit Name, Unit Abbreviation
Primary Key: Unit ID
Strong / Weak Entity: Strong
Fields to be indexed: Unit ID, Unit Name, Unit Abbreviation

Attributes:

Attribute Name	Unit ID	Unit Name	Unit Abbreviation
Description	Auto-increment integer that uniquely identifies a room.	Identifies and describes the purpose of the unit (Intensive Care, Medical/Surgical)	Short name for the unit (ICU, Med/Surg, NICU)
Domain / Type	Integer	String	String
Value / Range	0 – MaxID	Any	String length 10
Default Value	MaxID + 1	None	None
Null Value Allowed	No	No	No
Unique	Yes	Yes	Yes
Single or Multi-value	Single	Single	Single
Simple or Composite	Simple	Simple	Simple

Entity Name: Patient

Description: A patient is sent to a Unit depending on the severity of their medical situation. A Doctor will prescribe medication and/or order any Special need required. Periodically, a Nurse will assess them (take their heart rate, respiration rate and blood pressure) and document the results each time. Like employees, patients are stored with basic contact and identifying information, along with their language (in case a translator is needed).

Candidate Keys: Patient ID, SSN
Primary Key: Patient ID
Strong / Weak Entity: Strong
Fields to be indexed: Patient ID, Name, SSN, Phone Number, Insurance Plan

Attributes:

Attribute Name	Patient ID	SSN	Name	Address
Description	Number assigned by the hospital to each patient that uniquely identifies them. Can be implemented as an auto-increment integer.	Social Security Number; can be used to uniquely identify Employees.	Name of Employee (First, Middle, Last)	Street Address, City, State, Zip Code
Domain / Type	Integer	Integer	String (3 Strings)	String, String, String, Integer
Value / Range	0 - MaxID	000000000 - 999999999 (any 9-digit Integer)	Any	Any, Any, Any, 00000-99999
Default Value	MaxID + 1	None	None	None
Null Value Allowed	No	No	No	No
Unique	Yes	Yes	No	No
Single or Multi-value	Single	Single	Single	Single
Simple or Composite	Simple	Simple	Composite	Simple

Patient Attributes Continued:

Attribute Name	Phone Number	Birth Date	Gender	Insurance Plan
Description	Primary contact number represented as an Integer	Date the patient was born.	Single letter that identifies the gender of the patient	Medicare, uninsured, etc.
Domain / Type	Integer	Date / time	Character	String
Value / Range	0000000000 - 9999999999	Any	F / M	Any
Default Value	None	None	None	None
Null Value Allowed	No	No	No	No
Unique	No	No	No	No
Single or Multi-value	Single	Single	Single	Single
Simple or Composite	Simple	Simple	Simple	Simple

Patient Attributes Continued:

Attribute Name	Admitted Date	Discharged Date	Language Spoken
Description	The start date of a period of stay for the patient (a patient can stay multiple times)	The end date of a period of stay of a patient.	Identifies the primary language of the patient (indicating whether they need a translator)
Domain / Type	Date	Date	String
Value / Range	Any	Any	Any
Default Value	None	None	None
Null Value Allowed	No	Yes	No
Unique	No	No	No
Single or Multi-value	Multi-value	Multi-value	Single
Simple or Composite	Simple	Simple	Simple

1.2.2 Relationship Set Description

Relationships associate two or more entities of the same type. They are defined by which entities they relate, as well as additional attributes describing how entities are related. They also specify constraints that control how many entities are related to each other. Individual entities from different types are related to each other by participating in relationship instances.
We now define each relationship type: the entity types it relates, the constraints on cardinality and participation, and any additional attributes describing how entities in participating in the relationship set are related.

Relationship: prescribes
Description: Doctors at the hospital order patients to take medications with specific instructions (prescriptions) by prescribing. Prescriptions must be prescribed by a single, licensed doctor, and doctors order many prescriptions on different dates as they work at the hospital. When doctors prescribe medication, they detail instructions about how to administer the medication as well as the time over which the medication should be administered. However, most of this information is stored in the **Prescription** entity; in the ER model, relationship attributes can be shifted to the entity type on the "N" side of a 1..N relationship.
Entity Sets Involved: Doctor, Prescription
Mapping Cardinality: 1..N
Descriptive Field: None
Participation Constraint: Partial participation for Doctor. Total participation for Prescription.
All prescriptions must be ordered by a single doctor; a prescription is not valid unless a licensed doctor issues it. However, doctors can still exist without having yet prescribed medication.

Relationship: prescribed for
Description: Prescriptions are prescribed for patients. Each patient can receive many new prescriptions as their condition changes and doctors order new treatment. Each prescription is for a single patient, since each patient may need to receive medication with different amounts or administration methods (Multiple patients cannot share prescriptions).
Entity Sets Involved: Prescription, Patient
Mapping Cardinality: N..1
Descriptive Field: None
Participation Constraint: Total participation for Prescription. Partial participation for Patient.
Prescriptions are made for individual, specific patients; therefore, a prescription cannot exist without being related to a patient. Patients can be housed by the hospital without receiving prescriptions, so participation of prescription is partial.

Relationship: administers
Description: Nurses administer the medication described in a prescription. Many different nurses may share the responsibility of administering medicine on different days, so the relationship has cardinality "many to many." Nurses are implicitly required to administer medications at the right times based on the "Frequency" field contained in the prescription. The **administers** relationship, however, forces Nurses to record specifically when they give medication to ensure that they are following the doctor's orders correctly and not missing any times. Recording this data is legally required.
Entity Sets Involved: Nurse, Prescription
Mapping Cardinality: M..N
Descriptive Field: Date/time
Participation Constraint: Partial participation for both Nurse and Prescription.
When prescriptions are first prescribed by doctors, they have not been administered by nurses. Nurses can be employed at the hospital even if they have not yet administered a prescription.

Relationship: takes
Description: A single nurse will periodically perform a basic assessment of the patient's health (usually every four hours). One nurse performs the assessment, but nurses take many assessments throughout the day. It is important to record the date/time of each assessment to understand how the patient's condition is progressing over time.
Entity Sets Involved: Nurse, Assessment
Mapping Cardinality: 1..N
Descriptive Field: Date/time
Participation Constraint: Partial participation for Nurse. Total participation for Assessment.
An assessment is not valid unless it has the signature of a Nurse. Nurses can be employed at the hospital even if they have not made an assessment.

Relationship: undergoes
Description: A patient undergoes an assessment. A patient can undergo many assessments, but each assessment provides health information about only one patient. There are no descriptive fields, since the date/time of the assessment is included in another relationship.
Entity Sets Involved: Patient, Assessment
Mapping Cardinality: 1..N
Descriptive Field: None
Participation Constraint: Partial participation for Patient. Total participation for Assessment.
Newly admitted patients have not yet received assessments. However, an assessment (basic health information about a patient) is meaningless without the patient it describes; thus assessment participation is total.

Relationship: orders
Description: Each special need is ordered by a single doctor, and a doctor can order many special needs. The doctor orders a special need on a specific date.
Entity Sets Involved: Doctor, Special Need
Mapping Cardinality: 1..N
Descriptive Field: Date/time
Participation Constraint: Partial participation for Doctor. Total participation for Special Need.
Doctors can be employed without having ordered a special need (partial participation for doctor). However, each special need must be ordered by a (single) doctor, and the participation of special need is total.

Relationship: hasActivity
Description: Each Special Need has a single activity. The same activity can appear in multiple Special Need orders.
Entity Sets Involved: Special Need, Activity
Mapping Cardinality: N..1
Descriptive Field: None
Participation Constraint: Total participation for Special Need. Partial participation for Activity.
Special Needs each must be tied to an activity. An activity can exist if it isn't being used in a Special Need order.

Relationship: performs
Description: Nurses perform each of the special needs assigned to a patient at specific times throughout the day. Nurses are required to perform activities such that they are in compliance with the "Frequency" attribute of the special need as ordered by the doctor. This relationship forces nurses to record their activities and ensure they are not missing times. Many nurses can share responsibility for the same special need order over time, and individual nurses can perform many different special needs orders. Thus, the relationship is M..N.
Entity Sets Involved: Nurse, Special Need
Mapping Cardinality: M..N
Descriptive Field: Date/time
Participation Constraint: Partial participation for both Nurse and Special Need.
Nurses can be employed at the hospital without having performed a special need. Also, special needs have not yet been performed by nurses when they are initially ordered by doctors. Thus participation is partial for both nurses and special needs.

Relationship: receives
Description: A patient receives a special need order from a doctor. There are no descriptive fields since the date the patient receives the order is part of the **orders** relationship. Each special need order is specific to a single patient, but each patient can receive many individual special need orders.
Entity Sets Involved: Patient, Special Need
Mapping Cardinality: 1..N
Descriptive Field: None
Participation Constraint: Partial participation for Patient. Total participation for Special Need.
Newly admitted patients have not yet been given a special nee order. However, a special need order describes actions to be taken with a specific patient; thus special needs cannot exist without being related to patients.

Relationship: assigned to
Description: Nurses are assigned to a patient for the duration of their shift. This relationship allows hospital staff to keep track of which nurses should have performed activities and administered medication for a patient during a shift's time period. One nurse is assigned to one patient during a shift, but since our database stores historical data, the relationship is M..N; over the course of time, many patients may be assigned to the same nurse, and vice versa.
Entity Sets Involved: Nurse, Patient
Mapping Cardinality: M..N
Descriptive Field: Start Date/time, End Date/time
Participation Constraint: Partial participation for both Nurse and Patient.
Nurses can be employed at a hospital without being currently assigned to a patient. Nurse assignments exist only for the duration of a working shift, so patients can reside in the hospital without being yet assigned to a nurse.

Relationship: works in
Description: A nurse is contracted to work in a single unit for the duration of the contract. Since our model records historical data, the relationship is M..N. Over the course of time, a nurse might work for many units, and a unit may have several nurses employed.
Entity Sets Involved: Nurse, Unit
Mapping Cardinality: M..N
Descriptive Field: Start Date/time, End Date/time
Participation Constraint: Total participation for both Nurse and Unit.
Nurses' work contracts specify the units that nurses work for. Therefore, nurses cannot exist without being related to a unit. Also, all units have employed nurses.

Relationship: placed
Description: A patient is placed in (or in habits) a room. Nurses visit the room that a patient inhabits to administer treatment. At a specific point in time, one patient is placed in one room. However, patients can move from room to room, and a room may house many patients over time as patients come and leave. Thus, the relationship is M..N.
Entity Sets Involved: Patient, Room
Mapping Cardinality: M..N
Descriptive Field: Start Date/time, End Date/time
Participation Constraint: Total participation for Patient. Partial participation for Room.
Since the scope of our database only encompasses patients who are living in the hospital, patients cannot exist without being related to a room. Since new rooms can be empty with no patient history, rooms can exist without being related to a patient.

Relationship: belongs to
Description: A room is located inside of a unit. Unless the hospital changes its physical structure, rooms do not move between units. Thus a room has a single unit, and a unit contains many rooms.
Entity Sets Involved: Room, Unit
Mapping Cardinality: N..1
Descriptive Field: None
Participation Constraint: Total participation for both Room and Unit.
All rooms belong to a unit, and a unit must contain rooms. Thus both participations are total.

1.2.3 Related Entity Set

Specialization describes the process of forming subclasses of an entity type. Subclasses inherit all of the attributes of the entity type they are derived from (also called their "superclass"), while adding attributes that are specific to subclasses. In our database, Nurse and Doctor are subclasses of the Employee entity type.

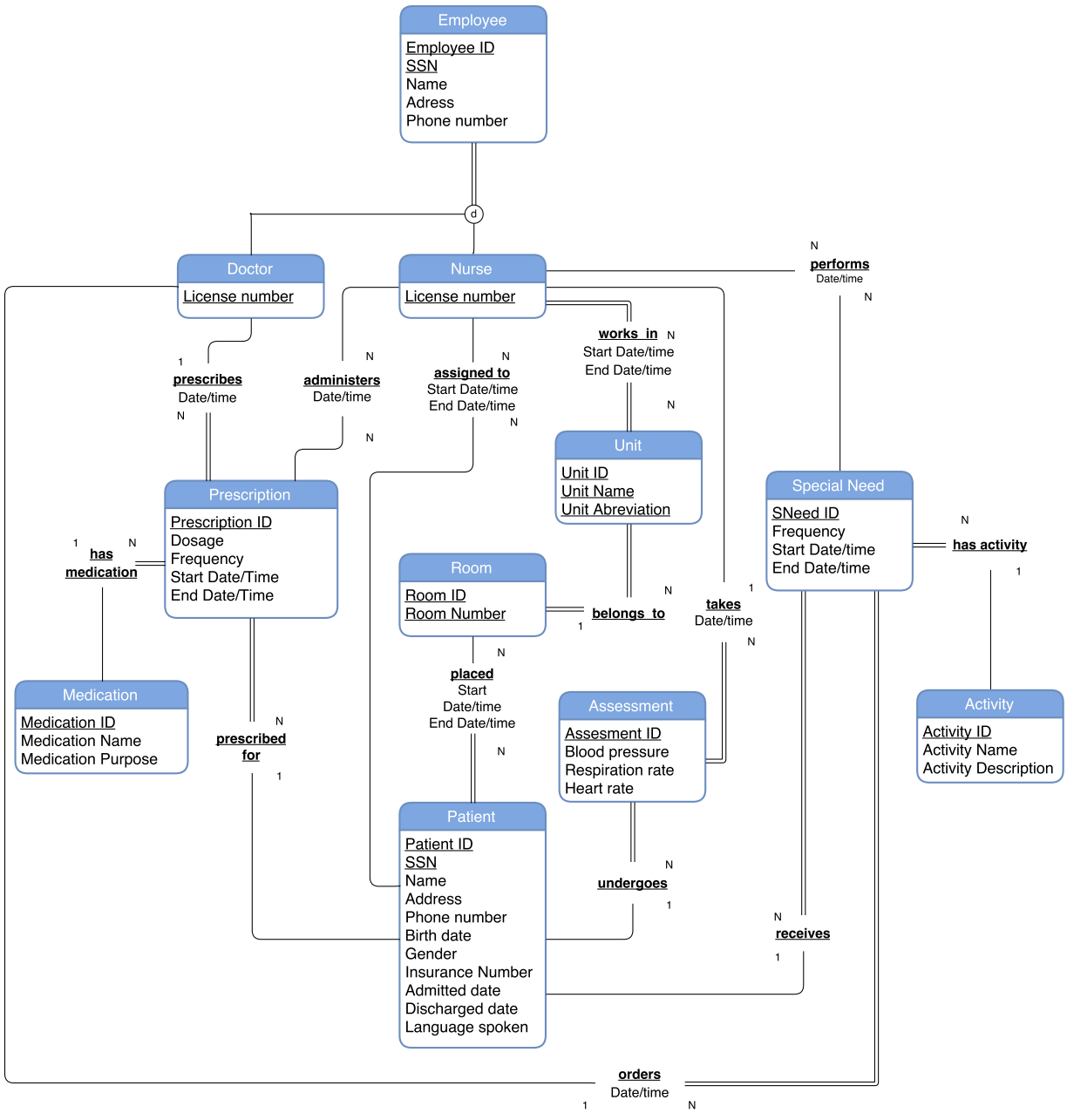
Generalization is the opposite of specialization; it involves uniting multiple related subclass entity types under a superclass. We formed the employee superclass in our conceptual database through generalization – we discovered that both doctors and nurses share much of the same information, and united them under the Employee superclass.

Specializations and generalizations have constraints on "completeness" and "disjointedness." The specialization of employee into nurse and doctor is **disjoint** because employees can never be both nurses and doctor at the same time. This is also called an "Is_A" specialization. The completeness constraint is **total**. This means that employees have to be either doctors or nurses. This constraint exists because the scope of our conceptual database excludes employees who aren't involved in patient care.

1.2.4 ER Diagram

To visualize how relationships associate different entities in an ER Model, it is useful to use an ER Diagram. In an ER diagram, relationships are represented as lines that connect participating entity types. Double lines are used to represent total participation, while single lines represent partial. "1" and "N" symbols indicate the cardinality of relationships. Entity types are represented as boxes which listings of their contained attributes.

The ER diagram for Charity General Hospital follows:


Employee
Employee ID
SSN
Name
Adress
Phone number
d
Doctor
License number
Nurse
License number
performs
Date/time
prescribes
Date/time
administers
Date/time
assigned to
Start Date/time
End Date/time
works_in
Start Date/time
End Date/time
Unit
Unit ID
Unit Name
Unit Abreviation
Special Need
SNeed ID
Frequency
Start Date/time
End Date/time
has activity
Prescription
Prescription ID
Dosage
Frequency
Start Date/Time
End Date/Time
has medication
Room
Room ID
Room Number
belongs_to
takes
Date/time
placed
Start
Date/time
End Date/time
Medication
Medication ID
Medication Name
Medication Purpose
prescribed for
Assessment
Assesment ID
Blood pressure
Respiration rate
Heart rate
Activity
Activity ID
Activity Name
Activity Description
Patient
Patient ID
SSN
Name
Address
Phone number
Birth date
Gender
Insurance Number
Admitted date
Discharged date
Language spoken
undergoes
receives
orders
Date/time

2. Conceptual Database and Logical Database

A conceptual database is used to easily visualize a database, while a logical database is used to represent how a database will be stored in a software implementation. Before we implement our database, we need to convert to a logical design. To move from our conceptual database design to a logical database design, we will convert the E-R model from the previous section to a relational model.

This section first describes the E-R model and the relational model, including their histories, purposes, and similarities and differences. Then, we will describe in detail the method for converting between the E-R model and the relational model. After we describe the process, we will implement it on our E-R model, and present the relational model for our database.

After creating the relational model, we will present sample data for our relational database. Then, we will create queries for the data using three formal querying languages: relational algebra, tuple relational calculus, and domain relational calculus.

2.1 E-R Model and Relational Model

The E-R model describes a conceptual database design, while a relational model describes a logical design. Here, we will describe both models and compare them.

2.1.1 Description of E-R Model and Relational Model

The E-R and relational database model each have a different histories, purpose, and set of defining features. The E-R model was developed by Peter Chen; he first introduced it in a 1976 publication. The E-R model is a method for creating conceptual database designs. It allows designers to represent the objects that make up a business or organization along with how they are related, and is best conveyed using a simple, easy-to-read diagram. The E-R model is high-level - it represents the structure of the database with a simple model that basic users, business owners, and technical designers can all understand.

The relational model was first introduced by E.F. Codd of IBM in 1970. The relational model is a method for creating logical database designs - it reflects how the database will be implemented in software. The relational model is rooted in discrete mathematics and set theory, and allows designers to represent the structure of a database using one basic building block - the “relation.” The relational model has been implemented by several commercial database management systems (DBMS), including Oracle, MySQL, and TSQL. The relational model is frequently grouped with formal querying languages: relational algebra, tuple relational calculus, and domain relational calculus.

The E-R model and relational model have different defining features. The E-R model uses two basic concepts: “entities” - which represent the objects that make up an organization - and “relationships” - which describe how different entities are associated with each other. Both are best represented using a diagram, where entities are boxes and relationships are lines that connect the boxes. Entities and relationships both have attributes that describe the information contained within, and relationships have constraints that explain how many times entities can be related to each other. Attributes can be grouped together so they are easier to understand.

The relational model is defined by only one concept - the “relation.” The relational model consists of tuples - single, flat lists of related values. Relational schema are lists of attributes that describe the purpose of each value in a tuple (and its domain), and all the tuples that belong to the same relational schema are grouped into a relation instance - essentially a table where attributes are columns and individual tuples are rows.

The main difference between the E-R model and the Relational model lies in their purpose. The purpose of E-R model is to visualize the structure of a database in an easy-to-understand way, so that business owners and technical designers can communicate and participate in the design of the database. It is too abstract and complicated to be implemented in software practically. The relational model, on the other hand, is designed to represent the database as it will be implemented in software. It is more fundamentally simple, but it is harder to understand the purpose or structure of a database by looking at the relational database schema (unlike with the ER-diagram). The relational model is rooted in mathematical theory, and is used with mathematical querying languages.

2.1.2 Comparison of Two Different Models

The E-R model has many advantages over the relational model. The E-R model is easier to understand and visually represents the database, so it is better for communicating ideas between software designers and less technologically savvy business users. The E-R model excludes the details of implementing the design in software, so it is easier to make large changes and is thus better for drafting ideas. The E-R model also has disadvantages. It is not based in mathematical theory, so it cannot be used with any formal mathematical querying languages. It is too complex and abstract, so no commercial DBMS use the E-R model. The E-R model is also not standardized - because it is so flexible, one can represent the same database design with many different ER-diagrams. This can create confusion between designers.

The relational model also has advantages and disadvantages. The relational model is formal and standardized - it is rooted in discrete mathematics, and formal, mathematical querying languages have been designed for it. The relational model is much more feasible for implementing software since it is more fundamentally simple; relation instances can be represented as tables, which are perfect for computer storage. The main disadvantage of the relational model is that it is much more difficult to understand - entities and relationships are blurred into the same building block, so it is difficult to

understand the purpose of each relation schema. Attributes cannot be grouped, so all simple components are listed fully, making for a dense read. The relational model should not be used to communicate with business users.

The E-R model and relational model have some similarities. Both models can be used to represent how data is structured. Like relation schema from the relational model, entity and relationship types from the E-R model include a name and a list of attributes. Both models have the concept of a "type" (or schema in the relational model) and "instance" (or tuple in the relational model) - the type describes what kind of data is included and its purpose, while the instance actually contains the data. Both models use some sort of constraints, or sets of rules, to describe how data is related.

The main difference between the E-R model and the relational model is that the E-R model uses two concepts - "entities" and "relationships" instead of one (the relational model only uses the "relation"). Also, entities and relationships in the E-R model can have complex, multi-value attributes, while attributes in the relational model must be atomic. The E-R model has different kinds of constraints than the relational model. The E-R model has cardinality and participation constraints on relationships that determine how many of each entity type in the relationship can be related to each other. The relational model has no cardinality or participation constraints, but has integrity constraints to enforce coherency between relations that reference each other.

2.2 Conversion of Conceptual Database Model to Logical Database Model

Now that both the E-R model and the relational model have been described, we will dive into a detailed explanation of the process for converting between the two. First, we will describe how to convert entity types, then relation types. Then, we will describe the process of building constraints to ensure the data in the relational model has integrity.

2.2.1 Converting Entity Types to Relations

When converting from an ER Model to a Relational Model, all *entity types* must be represented as a set of *relation schemas*. Each relation schema contains a list of attributes with single-value domains; ER Model Entity Types, however, are more complex, with composite and multi-value attributes, as well as weak entity types that have no key.

In this section, we will explain how to represent both weak and strong entity types as relations. Then, we explain how to represent both simple and composite entity attributes as relation attributes with atomic domains. Finally, we explain how to represent single and multiple-value entity attributes through relation attributes.

Converting Strong Entities into Relations

Each strong entity type E is converted into one relation schema R. The relation schema has the same name as the strong entity type. The attributes of the relation schema are the simple, single-value attributes of the entity type along with the simple components of the entity type's composite attributes (see **Mapping of Simple and Composite Attributes** and **Mapping of Single and Multi-Value Attributes** for more information on converting complex attributes). One of the key attributes of the entity type is made into the primary key attribute of the relation schema, while additional key attributes of the entity type become candidate keys of the relation schema (candidate keys are unique, but not used to identify tuples in a relation state).

Converting Weak Entities into Relations

Each weak entity is converted into a relation R that represents a strong entity. R has the same name as the weak entity. R contains the primary key of the relation for the weak entity's owner as a foreign key attribute (if the weak entity's owner is also weak entity type, the owner must first be converted into a strong entity so that its corresponding relation schema can have a primary key attribute). R also contains the weak entity type's partial key attribute(s) and other attributes. Together, the partial key attribute(s) and the foreign key attribute become a primary key of R. Since R can be viewed as a representation of a strong entity, all of the methods used to represent attributes of strong entity types can also now be used here.

Assuming that all relations represent strong entities, we can now represent the entities' attributes as relation attributes.

Mapping of Simple and Composite Attributes

There are separate processes for mapping the simple and composite attributes of an entity type. The simple attributes of an entity type become attributes of the corresponding relation schema. The simple components of the entity type's composite attributes each become separate attributes in the corresponding relation schema.

Mapping of Single and Multi-Value Attributes

There are separate processes for mapping the single and multi-value attributes of an entity type. The single-value attributes of an entity type simply become attributes of the relation schema R that corresponds to the entity type. The process for representing multi-valued attributes is more complex, however. Each multi-value attribute of an entity type is represented with a new, separate relation R_A. The attributes of R_A are the multi-value attribute itself, as well as the primary key of R as a foreign key attribute. The primary key of R_A is made up of the multi-value attribute and the foreign key attribute

combined. If the multi-value attribute is also composite, only the *unique* simple components of the attribute are part of the primary key of R_A.

2.2.2 Converting Relationship Types to Relations

In an ER Model, there are two main concepts - entities and relationships. However, the relational model only has one concept - the relation - so relationship types must be represented through relation schemas (either through separate relation schemas or attributes in relation schemas derived from entity types). In this section, we will explain how to represent relationship types which...

- ...have different cardinality constraints (1:1, 1:N, M:N)
- ...represent the "IsA" and "HasA" concepts of superclasses and subclasses have
- ...relationships to other relationship types
- ...are recursive (only involve one entity type)
- ...involve more than 2 entity types
- ...are category or union types

Mapping of Relationship Types with a 1:1 Cardinality Constraint

If a relationship type relates entity types A and B with a 1:1 cardinality constraint and A and B are converted into relations R_A and R_B, then each instance of R_A should be related to exactly one instance of R_B. There are three methods used to represent this constraint using relations:

1. **Foreign Key Approach**: In this approach, the primary key of R_A is made into a foreign key attribute of R_B (or vice versa). All simple (and simple components of composite) attributes that belong to the relationship type also become attributes of the relation with the foreign key.
2. **Merged Relation Approach**: In this approach, the attributes of R_A and R_B are combined into a single relation.
3. **Cross Reference Approach**: In this approach, a new relation R is created to represent the relationship type. R is referred to as a "relationship relation." R will contain the primary keys of R_A and R_B as foreign key attributes, as well as the simple (and simple components of composite) attributes of the relationship type. The primary key of R is one of the foreign keys.

Each approach has advantages and disadvantages.
The **Merged Relation Approach** is almost useless, because if two relations can be combined into one, then their corresponding entity types should have been combined when designing the conceptual database.
The **Foreign Key Approach** is useful because it decreases the number of join operations when querying, but should be used only if one of the entity types has *total* participation in the relationship - otherwise, the foreign key attribute will be NULL for relations that do not participate, and storage space will be wasted.
The **Cross Reference Approach** is useful if neither of the participating entity types have total participation, but increases the number of joins in queries.

Mapping of 1:N Relationship Types

If a relationship type relates entity types A and B with a 1:N cardinality constraint and A and B are converted into relations R_A and R_B, then each instance of R_A can be related to multiple instances of R_B, and each instance of R_B can be related to only one instance of R_A. There are two methods used to represent this constraint using relations:

1. **Foreign Key Approach**: The approach is the same for converting 1:1 relationship types, except the foreign key and relationship type attributes must belong to the relation derived from the entity on the "N" side of the relationship (in this case, R_B). This is because entities on the "N" side can only be related to the other entity type "1" time, so their participation in the relationship is unique.
2. **Cross Reference Approach**: The approach is the same for converting 1:1 relationship types, except the primary key of the relationship relation R must be the foreign key for the relation that represents the entity on the "N" side of the relationship.

The foreign key and cross reference approaches have the same advantages and disadvantages as as for 1:1 and 1:N relationship type mappings.

Mapping of M:N Relationship Types

If a relationship type S relates entity types A and B with a M:N cardinality constraint and A and B are converted into relations R_A and R_B, then each instance of R_A can be related to multiple instances of R_B, and each instance of R_B can also be related to multiple instances of R_A.

The only possible method for converting an M:N Relationship Type is through the **Cross Reference Approach**. A relationship relation R is created to represent the relationship type, and it contains the primary keys of R_A and R_B as foreign keys (as well as all of the simple and simple-component relationship type attributes). Both the foreign keys combined form the primary key for R.

Mapping of Superclasses and Subclasses for the "IsA" Relationship

The "IsA" Relationship occurs when entity types are *disjoint* subclasses of a superclass entity type. In other words, an entity belongs to no more than one subclass. There are three methods for representing this relationship with relations.

1. **Multiple relations - superclass and subclass:** In this approach, a superclass entity is given a relation R_{super}, and subclass entities are each given a relation R_{sub}. The superclass relation contains the attributes of the superclass entity. The subclass relation contains the attributes of the subclass entity, along with the primary key of the superclass relation as a foreign key attribute. The foreign key serves as the subclass relation's primary key.

2. **Multiple relations - subclass only:** In this approach, only subclass entities are given their own relations. The subclass relation contains the union of the attributes of the superclass entity type and the subclass entity type. If an entity belongs to multiple subclasses (overlapping), all corresponding subclass relations must share the same primary key values (this is why this approach works best for the "IsA" relationship, where there are no overlapping subclasses).
3. **Single relation with one type attribute:** In this approach, one relation R is created that contains the union of the attributes from the superclass and the attributes from *all* of the subclasses combined. R also contains a "type" attribute, which indicates which of the subclasses R belongs to.

Each approach has advantages and disadvantages. **Multiple relations - superclass and subclass** is useful because it works for every type of superclass/subclass relationship (disjoint "IsA" and overlapping "HasA", total and partial). However, because it creates a separate superclass relation, queries require more join operations.
Multiple relations - subclass only requires fewer join operations, but only works for relationships where the participation is total (an entity *must* belong to one of the subclasses).
Single relation with one type attribute requires the least join operations, but combining all the attributes creates a very large relation. Also, the attributes for the subclasses to which an entity does *not* belong will always be NULL. Therefore, the approach should only be used if the subclasses have very similar attributes.

Mapping of Superclasses and Subclasses for the "HasA" Relationship

The "HasA" Relationship occurs when entity types are *overlapping* subclasses of a superclass entity type. In other words, an entity can belong to multiple subclasses. There are two methods for representing this relationship with relations.

1. **Multiple relations - superclass and subclass:** The approach is the same as for the "IsA" superclass/subclass relationship.
2. **Single relation with multiple type attributes:** In this approach, one relation R is created that contains the union of the attributes from the superclass and the attributes from *all* of the subclasses combined. It also contains one boolean attribute for each of the possible subclasses. If the value for the subclass attribute is "true", then a relation instance tuple belongs to that subclass.

Each approach has advantages and disadvantages. **Multiple relations - superclass and subclass** has the same pros and cons as in the "IsA" superclass/subclass relationship type. **Single relation with multiple type attributes** requires the fewest number of joins, but the attributes for the subclasses to which an entity does *not* belong to will always be NULL, and storage space is thus wasted.

Mapping of Relationships to other Relationship Types

When mapping a relationship type that associates an entity (or a relationship type) with another relationship type, it is best to create a single primary key attribute for the relationship type. Then, the relationship between the relationship types can be mapped using the **Foreign Key Approach** or the **Cross Reference** approach depending on the cardinality of the relationship. The primary key of the *relationship type* will be used as a foreign key in either method.

Mapping of Recursive Relationships

Recursive relationships occur when one entity type (represented by relation R) is related to itself. There are two ways to represent this relationship:

1. **Foreign Key Approach**: In this approach, the R contains a foreign key attribute that references the primary key of the same relation R.
2. **Cross Reference Approach**: In this approach, a new relation $R_{recursive}$ is created to represent the recursive relationship. It contains two foreign keys that both reference the primary key of of R. The combination of the foreign keys forms the primary key of $R_{recursive}$.

Both approaches have advantages and disadvantages.
The **Foreign Key Approach** requires less join operations, but relation tuples corresponding to entity instances that do not participate in the relationship will have NULL values for the foreign key attribute. The **Cross Reference Approach** eliminates this problem but requires more join operations in queries.

Mapping of Relationships Between More than Two Entity Types

If a relationship type associates more than two entity types, the relationship type is represented as its own relationship relation R. Each entity type associated by the relationship type have already been converted into relations, and R contains the primary keys of these relations as foreign keys. The combination of the foreign keys forms the primary key of R (although some foreign keys may be excluded from the primary key if their corresponding relations represent entities on the “1” side of a 1:N cardinality constraint).

Mapping of Union Types (Categories)

A category or union type occurs when a relation for a subclass entity belongs to multiple superclass entities. Also, the relations corresponding to the different superclass entities have different primary keys. In this case, the superclass entities are each given a “surrogate key” attribute. If multiple entities are superclasses of the same subclass entity, the relation tuples corresponding to the superclass entities share the same value for the surrogate key.

2.2.3 Database Constraints

Database constraints ensure that all of the stored data is meaningful and coherent. The relational database model cannot accurately represent the ER model unless the tuples in the relation state satisfy certain rules and conditions. In any rule is violated, the relational database state is invalid, and the data is meaningless. In this section, we will detail each type of constraint, along with how they are enforced in a database management system (DBMS):

- Domain Constraints
- The Entity Constraint
- Primary Key and Unique Key Constraints
- Referential Constraints
- Check Constraints and Business Rules

Domain Constraints

Domain constraints ensure that the values for tuples in a relation state are within the domains of their corresponding attributes in the relation schema. Domain constraints include restricting the value for a given attribute to a specific data type, and to a subset of values within that data type. The DBMS can enforce domain constraints by rejecting any INSERT or UPDATE operations causing tuples to have invalid values, or by setting those values to NULL or "default."

The Entity Constraint

The entity constraint ensures that all tuples belonging to a relation state have a primary key value that is not NULL. It is important that the primary key value is not NULL so that each tuple can be uniquely identified. The DBMS enforces this constraint by rejecting (responding to with an error) INSERT and UPDATE operations where the primary key attribute is set to NULL.

Primary Key and Unique Key Constraints

All tuples belonging to a relation state must be uniquely identifiable. In other words, there is a minimal set of attributes - a "key" - whose values are never the same for any two tuples in the relation state. The Primary Key constraint ensures that no two tuples have the same values for primary key attributes. Some attributes in a relation schema can have a uniqueness constraint even if they are not the primary key.

In both cases, the DBMS enforces the constraints by rejecting INSERT and UPDATE operations where the primary key (or unique key) attribute value(s) of a tuple match those of any other existing tuple in the relation state. The DBMS can help enforce this constraint by auto-incrementing the primary key value of each new tuple during the INSERT operation if the attribute has an integer domain.

Referential Constraints

Some attributes of a relation schema are foreign key attributes; if a tuple in one relation state references a tuple in another using the foreign key, the referenced tuple *must exist*. The referential integrity constraint ensures that if relation R_1 contains a foreign key that references relation R_2 and t_1 is a tuple in relation state $r_1(R_1)$ then the foreign key value for t_1 matches the primary key value for some tuple t_2 in $r_2(R_2)$.

Referential constraints need to be enforced by the DBMS for INSERT, UPDATE, and DELETE operations. For the INSERT operation, any new tuple that has an invalid foreign key value is rejected or the foreign key value is set to NULL or "default" (only if possible). For the DELETE operation, there are three options:

1. **Restrict:** When deleting a tuple or referenced by the foreign key value of another tuple, reject the operation.
2. **Cascade:** When deleting a tuple, delete all tuples that reference the deleted tuple through a foreign key.
3. **Set Default**: When deleting a tuple, set the foreign key values for all other tuples which reference it to NULL or "default."

For the UPDATE operation, if the foreign key value becomes invalid, the operation is rejected or the foreign key value is set to NULL or "default." The three options for DELETE also apply to UPDATE when changing the primary key value of a tuple referenced by other tuples. The UPDATE operation can either be rejected (**restrict**), the foreign key values in all tuples referencing the primary key can be **cascade**-updated to the new primary key, or all foreign key values referencing the primary key can be **set to NULL** or "default."

Check Constraints and Business Rules

The above constraints ensure that all of the tuples in all relation states together form a valid relational database state - they ensure that the rules that define the relational model are satisfied. Some rules are specific to each business, however - these check constraints and business rules ensure that data fits users' expectations of how the business should run. These constraints must be custom-written by the database designer - they can either be enforced by code in applications that use the database, or through a constraint specification language provided by the DBMS, where "triggers" and "assertions" ensure no invalid data is being stored.

2.3 Convert Entity Relationship Model to Relational Model

Now that we have detailed the process for converting between a conceptual and logical database, we will apply the process to the conceptual design for Charity General Hospital. All of the entity types and relationship types must be converted into relational schema, and constraints must be created to preserve a valid relational database that fits the needs of the organization. We will also create sample tuples for each relation to how illustrate our logical database model will function in the real world.

2.3.1 Relational Schema for Logical Database

Each relation schema in the logical design will now be listed, along with its attributes, the entities and relationships it represents from the ER model, and all associated relational database constraints.

Relation Schema: employee
employee(**empID**, SSN, fName, mName, lName, street, city, state, zip, phone, licType, licNo)

Attributes:

*empID	integer, 0 – MaxID, primary key
SSN	integer, 000000000 – 999999999
fName	varchar2(255)
mName	varchar2(255)
lName	varchar2(255)
street	varchar2(255)
city	varchar2(255)
state	varchar2(2)
zip	integer, 00000 – 99999

employee Attributes Continued:

phone	integer, 0000000000 - 9999999999
licType	varchar(2), “MD” or “RN”
licNo	integer, 000000 - 999999

Candidate Keys: empID (primary), SSN

Primary Key/Entity Integrity Constraint: “empID” must be unique and cannot be NULL

Uniqueness Constraint: “SSN” must be unique

Business Constraint: None

Derivation From Entity and Relationship Types: Employee, Doctor, Nurse

Derived from the Employee entity type. Represents the Nurse and Doctor “is_a” specialization using the **single relation with one type attribute** method; “licType” is the type attribute for discerning between Nurse and Doctor. The composite “Name” and “Address” attributes of the Employee Entity Type are broken into their simple components.

Relation Schema: prescription
prescription(**rxID**, **empID**, **patientID**, **medID**, dosage, freq, sDate, eDate, rxDate)

Attributes:

*rxID	integer, 0 – MaxID, primary key
*empID	integer, 0 – MaxID, foreign key
*patientID	integer, 0 – MaxID, foreign key
*medID	integer, 0 – MaxID, foreign key
dosage	varchar2(255)
freq	varchar2(255)
sDate	date
eDate	date
rxDate	date

Candidate Keys: rxID (primary)

Primary Key/Entity Integrity Constraint: "rxID" is unique and cannot be NULL

Referential Integrity Constraint: "empID" is a foreign key for employee, "patientID" is a foreign key for patient, "medID" is a foreign key for medication

Business Constraint: "sDate" is never greater than "eDate"
The employee tuple referenced by "empID" must be for a doctor ("licType" = "MD")

Derivation From Entity and Relationship Types: Prescription, Prescription *has medication* Medication, Doctor *prescribes* Prescription, Prescription *prescribed for* Patient

Derived from the Prescription entity type. Represents the N:1 relationships to Doctor, Patient, and Medication using the **foreign key approach**, since the participation of Prescription is total for all of these relationships.

Relation Schema: medication
medication(**medID**, medName, medPurpose)

Attributes:

*medID	integer, 0 – MaxID, primary key
medName	varchar2(255)
medPurpose	varchar2(255)

Candidate Keys: medID, medName

Primary Key/Entity Integrity Constraint: "medID" is unique and cannot be NULL

Uniqueness Constraint: "medName" must be unique

Business Constraint: None

Derivation From Entity and Relationship Types: Medication

Derived from the Medication entity type. All attributes are simple and single-value.

Relation Schema: room
room(**roomID**, **unitID**, roomNo)

Attributes:

*roomID	integer, 0 – MaxID, primary key
*unitID	integer, 0 – MaxID, foreign key
roomNo	integer, 0 - 999

Candidate Keys: roomID (primary), [unitID, roomNo]
*[<key1>,<key2>] denotes a compound key

Primary Key/Entity Integrity Constraint: "roomID" is unique and cannot be NULL

Uniqueness Constraint: The combination of "unitID, roomNo" is unique (no two rooms in the same unit have the same number)

Referential Integrity Constraint: "unitID" is a foreign key for unit

Business Constraint: None

Derivation From Entity and Relationship Types: Room, Room *belongs to* Unit

Derived from the Room entity type. Represents the N:1 relationship with Unit using the **foreign key approach** since the participation of Room is total.

Relation Schema: unit
unit(**unitID**, unitName, unitAbbrv)

Attributes:

*unitID	integer, 0 – MaxID, primary key
unitName	varchar2(255)
unitAbbrv	varchar2(10)

Candidate Keys: unitID (primary), unitName, unitAbbrv

Primary Key/Entity Integrity Constraint: "unitID" is unique and cannot be NULL

Uniqueness Constraint: "unitName" is unique, "unitAbbrv" is unique

Business Constraint: None

Derivation From Entity and Relationship Types: Unit

Derived from the unit entity type. All attributes are simple and single-value.

Relation Schema: patient
patient(**patientID**, SSN, fName, mName, lName, street, city, state, zip, phone, dob, gender, insuranceStatus, language)

Attributes:

*patientID	integer, 0 – MaxID, primary key
SSN	integer, 000000000 – 999999999
fName	varchar2(255)
mName	varchar2(255)
lName	varchar2(255)
street	varchar2(255
city	vharchar2(255)
state	varchar2(255)
zip	integer, 00000 – 99999
phone	integer, 0000000000 – 9999999999
dob	date
gender	character, F or M
insuranceStatus	varchar2(255)
language	varchar2(255)

Candidate Keys: patientID, SSN

Primary Key/Entity Integrity Constraint: “patientID” is unique and cannot be NULL

Uniqueness Constraint: “SSN” is unique

Business Constraint: None

Derivation From Entity and Relationship Types: Patient

Derived from the Patient entity type. The composite “Name” and “Address” attributes are broken into simple component attributes.

Relation Schema: patientAdmitted
patientAdmitted(**patientID**, admittedDate, dischargedDate)

Attributes:

*patientID	integer, 0 – MaxID, foreign key
admittedDate	date
dischargedDate	date

Candidate Keys: [patientID, admittedDate]

Primary Key/Entity Integrity Constraint: The combination of “patientID, admittedDate” is unique and no fields can be NULL

Referential Integrity Constraint: “patientID” is a foreign key for patient

Business Constraint: “admittedDate” cannot be greater than “dischargedDate”

Derivation From Entity and Relationship Types: Patient
Derived from the multi-value attributes “Date Admitted” and “Date Discharged” in the Patient entity type.

Relation Schema: placed
placed(**patientID**, **roomID**, sDate, eDate)

Attributes:

*patientID	integer, 0 – MaxID, foreign key
*roomID	integer, 0 – MaxID, foreign key
sDate	date
eDate	date

Candidate Keys: [patientID, roomID, sDate]

Primary Key/Entity Integrity Constraint: The combination of "patientID, roomID, sDate" is unique and cannot be NULL

Referential Integrity Constraint: "roomID" is a foreign key for room

Business Constraint:
"sDate" cannot be greater than "eDate"

Derivation From Entity and Relationship Types: Patient *placed* Room

Derived from the M:N Patient *placed* Room relationship type using the **cross-reference approach**; placed is a "relationship relation" with foreign keys for Patient and Room.

Relation Schema: assessment
assessment(**asmtID**, **empID**, **patientID**, bpSystolic, bpDiastolic, respRate, heartRate, date)

Attributes:

*asmtID	integer, 0 – MaxID, primary key
*empID	integer, 0 – MaxID, foreign key
*patientID	integer, 0 – MaxID, foreign key
bpSystolic	integer, 0 – 300
bpDiastolic	integer, 0 – 300
respRate	integer, 0 – 300
heartRate	integer, 0 – 300
date	timestamp

Candidate Keys: asmtID, [empID, date], [patientID, date]

Primary Key/Entity Integrity Constraint: "asmtID" is unique and cannot be NULL

Referential Integrity Constraint: "empID" is a foreign key for employee, "patientID" is a foreign key for patient

Business Constraint: None

Derivation From Entity and Relationship Types: Assessment, Nurse *takes* Assessment, Patient *undergoes* Assessment

Derived the Assessment entity type. The composite attribute "Blood Pressure" is broken down into simple component attributes. Represents the N:1 relationships to Nurse and Patient using the **foreign key** approach, since the participation of Assessment in both relationships is total.

Relation Schema: specialNeed
specialNeed(**snID**, **empID**, **patientID**, **activityID**, freq, sDate, eDate, date)

Attributes:

*snID	integer, 0 – MaxID, primary key
*empID	integer, 0 – MaxID, foreign key
*patientID	integer, 0 – MaxID, foreign key
*activityID	integer, 0 – MaxID, foreign key
freq	varchar2(255)
sDate	date
eDate	date
date	date

Candidate Keys: snID

Primary Key/Entity Integrity Constraint: "snID" is unique and cannot be NULL

Referential Integrity Constraints: "empID" is a foreign key for employee, "patientID" is a foreign key for patient, "activityID" is a foreign key for activity

Business Constraints:
"sDate" cannot be greater than "eDate"
The employee tuple referenced by "empID" must be a doctor ("licType" = "MD")

Derivation From Entity and Relationship Types: Special Need, Special Need *has activity* Activity, Doctor *orders* Special Need

Derived from the Special Need entity type. All attributes are simple and single-value. Represents the N:1 relationships with Doctor and Activity using the **foreign key approach**, since the participation of Special Need in both relationships is total.

Relation Schema: activity
activity(**activityID**, activityName, desc)

Attributes:

*activityID	integer, 0 – MaxID, primary key
activityName	varchar2(255)
desc	varchar2(255)

Candidate Keys: activityID

Primary Key/Entity Integrity Constraint: "activityID" is unique and cannot be NULL

Uniqueness Constraint: "activityName" is unique

Business Constraint: None

Derivation From Entity and Relationship Types: Activity

Derived from the Activity entity type. All attributes are simple and single-value.

Relation Schema: administers
administers(**empID**, **rxID**, date)

Attributes:

*empID	integer, 0 – MaxID, foreign key
*rxID	integer, 0 – MaxID, foreign key
date	timestamp

Candidate Keys: [empID, date], [rxID, date]

Primary Key/Entity Integrity Constraint: The combination of "empID, date" must be unique and cannot be NULL

Business Constraint: The employee tuple referenced by "empID" is a nurse ("licType" = "RN")

Referential Integrity Constraints: "empID" is a foreign key for employee, "rxID" is a foreign key for prescription

Derivation From Entity and Relationship Types: Nurse *administers* Prescription

Derived from the M:N relationship between Nurse and Prescription using the **cross-reference approach**; administers is a "relationship relation."

Relation Schema: assignedTo
assignedTo(**empID**, **patientID**, sDate, eDate)

Attributes:

*empID	integer, 0 – MaxID, foreign key
*patientID	integer, 0 – MaxID, foreign key
sDate	timestamp
eDate	timestamp

Candidate Keys: [empID, patientID, sDate]

Primary Key/Entity Integrity Constraint: The combination of "empID, patientID, sDate" must be unique and cannot be NULL

Referential Integrity Constraints: "empID" is a foreign key for employee, "patientID" is a foreign key for patient

Business Constraint: "sDate" cannot be greater than "eDate"
The employee tuple referenced by "empID" is a nurse ("licType" = "RN")

Derivation From Entity and Relationship Types: Nurse *assigned to* Patient

Derived from the M:N relationship between Nurse and Patient using the **cross-reference approach**; assignedTo is a "relationship relation."

Relation Schema: performs
performs(**empID**, **snID**, date)

Attributes:

*empID	integer, 0 – MaxID, foreign key
*snID	integer, 0 – MaxID, foreign key
date	timestamp

Candidate Keys: [empID, date], [snID, date]

Primary Key/Entity Integrity Constraint: The combination of "empID, date" must be unique and cannot be NULL

Referential Integrity Constraints: "empID" is a foreign key for employee, "snID" is a foreign key for specialNeed

Business Constraint: "sDate" cannot be greater than "eDate"
The employee tuple referenced by "empID" is a nurse ("licType" = "RN")

Derivation From Entity and Relationship Types: Nurse *performs* Special Need

Derived from the M:N relationship between Nurse and Special Need using the **cross-reference approach**; performs is a "relationship relation."

Relation Schema: worksIn
worksIn(**empID, unitID**, sDate, eDate)

Attributes:

*empID	integer, 0 – MaxID, foreign key
*unitID	integer, 0 – MaxID, foreign key
sDate	date
eDate	date

Candidate Keys: [empID, unitID, sDate]

Primary Key/Entity Integrity Constraint: The combination of "empID, unitID, sDate" must be unique and cannot be NULL

Referential Integrity Constraints: "empID" is a foreign key for employee, "unitID" is a foreign key for unit

Business Constraints: "sDate" cannot be greater than "eDate"
The employee tuple referenced by "empID" is a nurse ("licType" = "RN")

Derivation From Entity and Relationship Types: Nurse *works in* Unit

Derived from the M:N relationship between Nurse and Unit using the **cross-reference approach**; worksIn is a "relationship relation."

2.3.2 Sample Data of Relation

We will now list tuples that belong to sample relation states for each relational schema in the relational database schema; the tuples will form a valid relational database state. The tuples will be listed in a table format, were relational schema attributes are columns and individual tuples are rows. Each relation derived from an entity will be given between 10 and 20 sample tuples, while all other relations will be given between 60 and 100.

Employee											
empID	SSN	fName	mName	lName	street	city	state	zip	phone	licType	licNo
1	881076941	Albert	Terry	Perry	822 Rowland Parkway	Korkino	California	87689	3215391654	RN	603812
2	531711882	Arthur	Willie	Moore	8036 Rigney Point	Cambas	Texas	92127	2929343205	RN	365620
3	118454889	Ruby	Doris	Reid	71112 Grasskamp Junction	Pattani	Colorado	52743	4655925227	RN	261701
4	171400011	Deborah	Louis	Gordon	917 Golf Course Trail	Makato	Mississippi	26234	9451883271	RN	396551
5	792092438	Jose	Louis	Day	31739 Fremont Pass	Washington	North Carolina	13524	9060612037	RN	342062
6	485255265	Stephanie	Anne	Johnson	16849 Ramsey Alley	Alukama	Oregon	84370	9843948439	RN	328804
7	777181715	William	Bentley	Price	1827 Michigan Point	Barbacena	Washington	89349	5106048310	RN	980387
8	158853851	Ray	William	Wells	9527 Cody Terrace	Princeton	Colorado	51324	8773206927	RN	916003
9	763896972	Joyce	Maria	Price	293 Merchant Terrace	Santo Amaro	Florida	27256	2954554242	RN	809919
10	959716420	Mark	Allen	Jefferson	5030 Hagan Crossing	Nashville	Texas	92723	3102260136	RN	543560
11	669336025	Samuel	Mich	Taylor	7730 Scoville Drive	Francisco Villa	California	55310	1230115500	MD	240773
12	916527281	David	Brian	Madson	8508 Starling Crossing	Fresno	California	63509	6807569285	MD	669696
13	321693401	Jose	Walter	Hall	296 Upham Plaza	Grand Rapids	Texas	77201	7274041819	MD	901056
14	321083187	Harold	Lawrence	Roberts	5041 Michigan Way	Antigonish	Colorado	38103	7396402185	MD	286169
15	907309890	Ruth	Stephanie	Hamilton	6283 Magdeline Place	Brooklyn	Pennsylvania	88929	9165982423	MD	926923

Patient													
patientID	SSN	fName	mName	lName	street	city	state	zip	phone	dob	gender	insuranceStatus	language
1	465170825	Chloe	Terry	Price	1354 Valley Edge Alley	Creil	Maryland	31231	7637540760	9/28/2015	F	Medicare	English
2	647780742	Max	Catherine	Caulfield	1035 Dapin Terrace	Pirca	North Carolina	65576	7285187157	12/14/2015	F	Medicare	French
3	854015608	Kate	Juan	Marsh	9779 Stephen Pass	Gaofeng	New York	46774	6374340268	7/25/2015	F	Medicaid	English
4	672636578	Lisa	The	Plant	789 Straubel Pass	Memphis	Pennsylvania	95632	6489663918	9/2/2015	M	None	Spanish
5	169531477	William	George	Price	79658 Mockingbird Court	Pandean	Colorado	34807	2993860306	12/1/2015	M	None	English
6	504794456	Warren	Nicole	Graham	145 Blaine Street	Saint-Pie	Delaware	18044	1412685790	6/24/2015	M	Medicaid	Japaneese
7	264625346	Victoria	Ruby	Chase	76282 Northview Court	Ängelholm	Georgia	52169	6776861920	2/12/2015	F	Medicare	Spanish
8	960041017	Dana	Wayne	Ward	185 Bay Park	Santa Gertrudes	California	84421	4361094869	4/2/2015	F	Medicaid	Japaneese
9	441380586	Alyssa	Jeffrey	Anderson	22718 Fulton Parkway	Salitral	Arizona	20448	1899578244	1/12/2016	F	Medicare	Spanish
10	499757261	Rachel	Ernest	Amber	942 Stang Court	Ravenstone	California	96853	8836628248	5/5/2015	F	None	Japaneese
11	128458573	Frank	Jonathan	Bowers	5852 Kipling Drive	Doucard	Florida	20910	4601547204	3/11/2015	M	Medicare	Spanish
12	914594909	Stella	Rose	Hill	1804 Ridgeview Circle	Pencilrock	Pennsylvania	51083	7761872373	3/10/2015	F	None	Spanish
13	371083748	Pompidou	The	Dog	5342 Claremont Trail	Papercisors	Tennessee	97957	3085107873	1/17/2016	M	Medicare	Bark
14	196547325	Luke	Thomas	Parker	325 Jackson Point	Tulip	Connecticut	55423	4979614660	2/24/2015	F	Medicaid	Japaneese
15	878345785	Daniel	Sara	DaCosta	857 Farmco Junction	Pertination	California	17043	2914710251	2/5/2016	M	Medicaid	French

Room		
roomID	unitID	roomNo
1	1	101
2	1	102
3	1	103
4	1	104
5	1	105
6	1	106
7	1	107
8	2	108
9	2	109
10	2	110
11	2	201
12	2	202
13	2	203
14	2	204
15	3	205
16	3	206
17	3	207
18	3	208
19	3	209
20	3	210
21	4	301
22	4	302
23	4	303
24	4	304
25	4	305
26	5	306
27	5	307
28	5	308
29	5	309
30	5	310

Activity		
activityID	activityName	activityDescription
1	bed turn	Rotate patient in bed
2	wound care	Clean and dress wounds
3	catheter care	Change out catheter
4	showering	Assist patient with shower
5	burn care	Clean and care for burn wounds
6	physical therapy	Perform physical therapy for patient
7	functional mobility	Help patient walk and move about
8	assisted feeding	Help patient consume food
9	assisted toilet hygiene	Help patient use the bathroom
10	bathing	Assist patient with bath

Unit		
unitID	unitName	unitAbbrv
1	Accute assessment	ACU
2	Intensive care	ICU
3	Nenoatal intensive care	NICU
4	Physical therapy	PT
5	Medical/Surgical	MedSurg

Medication		
medID	medName	medPurpose
1	Triclosan	Painkiller
2	Acetaminophen	Blood thinner
3	Cytarabine	Insomnia
4	Flurazepam Hydrochloride	Laxative
5	Lisinopril	Antibiotic
6	Loratadine	Antibiotic
7	Ciprofloxacin	Blood thinner
8	Hydrocodon	Painkiller
9	Escitalopram Oxalate	Blood thinner
10	Spironolactone	Anti-allergin

Placed			
patientID	roomID	sDate	eDate
3	1	12/21/2015	12/31/2015
8	2	5/25/2015	6/11/2015
6	18	1/9/2016	2/1/2016
2	12	9/18/2015	10/15/2015
4	17	1/11/2016	2/10/2016
13	25	4/29/2015	5/17/2015
1	19	1/1/2016	1/20/2016
11	26	2/15/2015	3/1/2015
12	27	12/28/2015	1/16/2016
7	2	10/16/2015	11/15/2015
1	25	3/4/2015	3/17/2015
14	25	7/26/2015	8/6/2015
14	21	3/24/2015	4/16/2015
12	21	9/23/2015	10/23/2015
8	15	8/21/2015	9/15/2015
10	2	3/13/2015	3/30/2015
5	23	7/1/2015	7/28/2015
2	14	3/7/2015	4/5/2015
12	17	7/17/2015	8/2/2015
1	7	5/1/2015	5/19/2015
10	4	5/8/2015	5/29/2015
13	29	3/5/2015	3/24/2015
15	3	1/4/2016	1/25/2016
14	28	5/20/2015	6/1/2015
14	17	6/25/2015	7/24/2015
12	13	2/13/2015	3/1/2015
6	23	8/1/2015	8/28/2015
2	6	2/15/2015	3/5/2015
11	18	9/29/2015	10/11/2015
5	11	8/19/2015	9/10/2015
1	12	4/16/2015	5/7/2015
5	15	7/3/2015	8/1/2015
11	21	2/21/2015	3/7/2015
13	12	4/16/2015	5/13/2015
15	28	12/14/2015	1/11/2016
11	6	12/23/2015	1/21/2016
10	29	10/9/2015	11/5/2015
13	8	7/20/2015	8/10/2015
5	12	12/16/2015	1/13/2016

Patient Admitted		
patientID	admittedDate	dischargedDate
6	12/18/2015	12/18/2015
8	5/22/2015	5/22/2015
3	1/7/2016	1/7/2016
14	9/16/2015	9/16/2015
2	1/8/2016	1/8/2016
12	4/27/2015	4/27/2015
15	12/31/2015	12/31/2015
15	2/13/2015	2/13/2015
11	12/27/2015	12/27/2015
3	10/15/2015	10/15/2015
15	3/2/2015	3/2/2015
5	7/23/2015	7/23/2015
3	3/22/2015	3/22/2015
15	9/22/2015	9/22/2015
11	8/20/2015	8/20/2015
14	3/12/2015	3/12/2015
15	6/28/2015	6/28/2015
13	3/4/2015	3/4/2015
6	7/14/2015	7/14/2015
15	4/28/2015	4/28/2015
2	5/6/2015	5/6/2015
13	3/3/2015	3/3/2015
8	1/1/2016	1/1/2016
10	5/19/2015	5/19/2015
10	6/23/2015	6/23/2015
14	2/12/2015	2/12/2015
6	7/30/2015	7/30/2015
4	2/13/2015	2/13/2015
1	9/28/2015	9/28/2015
2	8/17/2015	8/17/2015
2	4/15/2015	4/15/2015
15	6/30/2015	6/30/2015
9	2/18/2015	2/18/2015
13	4/14/2015	4/14/2015
11	12/12/2015	12/12/2015
10	12/22/2015	12/22/2015
10	10/7/2015	10/7/2015
10	7/19/2015	7/19/2015
9	12/15/2015	12/15/2015
3	7/3/2015	7/3/2015

6	14	7/5/2015	7/26/2015
7	12	9/10/2015	10/3/2015
8	8	4/4/2015	4/24/2015
2	28	9/18/2015	10/14/2015
5	20	5/6/2015	6/5/2015
11	18	1/5/2016	1/29/2016
9	22	8/21/2015	9/10/2015
3	14	5/17/2015	5/31/2015
13	22	4/21/2015	5/9/2015
4	25	5/8/2015	5/28/2015
4	17	9/1/2015	9/22/2015
7	1	6/16/2015	7/11/2015
3	20	4/13/2015	4/24/2015
5	20	10/5/2015	11/4/2015
14	9	10/14/2015	11/3/2015
11	3	11/9/2015	12/4/2015
6	11	1/28/2016	2/9/2016
6	23	10/30/2015	11/29/2015
2	29	5/28/2015	6/22/2015
15	13	2/23/2015	3/12/2015
8	30	6/22/2015	7/5/2015
4	7	11/15/2015	12/6/2015
5	12	3/1/2015	3/23/2015
11	23	1/1/2016	1/18/2016
6	2	3/2/2015	3/14/2015
6	15	2/13/2015	3/9/2015
11	13	9/9/2015	9/25/2015
1	28	12/3/2015	12/13/2015
13	20	5/26/2015	6/16/2015
11	26	7/21/2015	7/31/2015
2	25	8/9/2015	9/5/2015
2	22	5/13/2015	5/25/2015
5	24	1/31/2016	2/13/2016
9	11	12/10/2015	12/25/2015
8	18	9/21/2015	10/16/2015
13	27	10/5/2015	11/3/2015
9	29	1/5/2016	1/31/2016
6	24	4/1/2015	4/30/2015
10	5	6/10/2015	6/28/2015
11	16	10/14/2015	10/31/2015
8	29	4/12/2015	4/23/2015

3	9/8/2015	9/8/2015
3	4/3/2015	4/3/2015
14	9/15/2015	9/15/2015
13	5/3/2015	5/3/2015
2	1/3/2016	1/3/2016
13	8/19/2015	8/19/2015
6	5/14/2015	5/14/2015
4	4/19/2015	4/19/2015
4	5/7/2015	5/7/2015
3	8/30/2015	8/30/2015
2	6/14/2015	6/14/2015
3	4/11/2015	4/11/2015
5	10/4/2015	10/4/2015
11	10/12/2015	10/12/2015
8	11/8/2015	11/8/2015
10	1/27/2016	1/27/2016
3	10/28/2015	10/28/2015
14	5/26/2015	5/26/2015
11	2/22/2015	2/22/2015
9	6/19/2015	6/19/2015
5	11/13/2015	11/13/2015
14	2/28/2015	2/28/2015
7	12/30/2015	12/30/2015
5	2/27/2015	2/27/2015
4	2/11/2015	2/11/2015
3	9/6/2015	9/6/2015
11	12/2/2015	12/2/2015
9	5/23/2015	5/23/2015
14	7/20/2015	7/20/2015
12	8/6/2015	8/6/2015
3	5/11/2015	5/11/2015
13	1/30/2016	1/30/2016
1	12/8/2015	12/8/2015
6	9/19/2015	9/19/2015
4	10/4/2015	10/4/2015
3	1/2/2016	1/2/2016
11	3/30/2015	3/30/2015
1	6/9/2015	6/9/2015
12	10/12/2015	10/12/2015
2	4/11/2015	4/11/2015

Prescription								
rxID	empID	patientID	medID	dosage	freq	sDate	eDate	rxDate
1	12	12	3	113 mg	Three times daily	8/22/2014	9/27/2014	8/21/2014
2	13	3	4	109 mg	Once daily	7/8/2015	8/14/2015	7/7/2015
3	13	14	4	83 mg	Three times daily	8/21/2014	9/30/2014	8/19/2014
4	13	14	10	177 mg	Three times daily	9/25/2014	11/1/2014	9/23/2014
5	13	1	4	168 mg	Once daily	2/3/2016	2/23/2016	2/1/2016
6	15	6	7	105 mg	Once daily	10/18/2014	12/2/2014	10/14/2014
7	14	12	10	100 mg	Three times daily	7/8/2014	8/20/2014	7/4/2014
8	11	13	1	142 mg	Once daily	5/15/2015	6/17/2015	5/10/2015
9	15	4	4	55 mg	Twice daily	12/29/2014	2/16/2015	12/27/2014
10	11	7	2	56 mg	Twice daily	6/30/2014	8/26/2014	6/25/2014
11	13	1	4	39 mg	Three times daily	1/28/2015	3/24/2015	1/23/2015
12	12	13	8	85 mg	Once daily	1/14/2015	3/13/2015	1/13/2015
13	14	6	3	93 mg	Twice daily	4/24/2015	6/17/2015	4/20/2015
14	12	4	5	24 mg	Twice daily	8/4/2015	9/17/2015	8/3/2015
15	15	1	4	35 mg	Once daily	7/29/2014	9/23/2014	7/25/2014
16	13	12	5	136 mg	Once daily	6/7/2014	8/5/2014	6/4/2014
17	13	9	4	122 mg	Three times daily	4/23/2015	5/24/2015	4/20/2015
18	15	4	3	65 mg	Three times daily	1/24/2015	3/9/2015	1/22/2015
19	13	13	2	88 mg	Twice daily	2/12/2014	3/7/2014	2/11/2014
20	14	3	2	52 mg	Three times daily	8/22/2014	10/17/2014	8/18/2014

Administers		
empID	rxID	date
2	6	12/8/2015 17:27:51
5	19	3/30/2015 20:40:58
8	15	3/19/2015 5:55:04
7	15	2/28/2015 17:34:47
7	11	6/20/2015 15:53:07
5	14	7/21/2015 22:47:28
8	8	11/25/2015 15:50:01
2	14	1/26/2016 19:02:34
7	13	9/8/2015 19:51:07
7	19	11/17/2015 5:39:47
5	3	6/26/2015 19:48:40
5	15	4/9/2015 12:00:33
1	3	8/18/2015 7:02:25
9	7	4/12/2015 20:14:42
5	4	2/9/2016 5:54:53
5	12	2/11/2016 14:03:24
10	1	2/22/2015 16:19:47
1	18	6/29/2015 17:34:31
9	7	2/19/2015 18:17:36
8	5	10/21/2015 20:05:13
10	6	2/16/2015 4:59:07
10	7	11/25/2015 21:14:34
8	6	2/16/2015 7:58:25
8	17	6/21/2015 0:10:34
9	5	7/10/2015 13:02:53
7	12	3/1/2015 22:12:44
6	9	7/8/2015 8:33:31
2	14	3/8/2015 7:40:16
5	3	4/18/2015 1:10:35
1	2	6/21/2015 15:27:58
4	18	9/9/2015 4:12:37
10	10	3/30/2015 12:18:03
3	17	11/19/2015 14:21:26
3	3	2/24/2015 13:49:18
6	11	12/11/2015 14:01:04
5	11	6/30/2015 21:24:33
2	9	11/6/2015 16:15:34
1	15	6/10/2015 21:46:01
6	6	5/25/2015 0:10:58
8	6	3/22/2015 22:26:11
1	5	3/7/2015 23:59:21
1	1	11/22/2015 14:26:44

Assigned To			
empID	patientID	sDate	eDate
9	10	2/10/2016 18:56:00	10/5/2015 0:37:00
5	1	10/4/2015 12:37:00	12/7/2015 12:07:00
2	2	12/7/2015 0:07:00	7/1/2015 9:47:00
5	2	6/30/2015 21:47:00	1/13/2016 13:39:00
10	2	1/13/2016 1:39:00	2/17/2015 9:47:00
10	5	2/16/2015 21:47:00	7/3/2015 5:09:00
4	7	7/2/2015 17:09:00	10/9/2015 2:04:00
1	2	10/8/2015 14:04:00	1/4/2016 8:00:00
1	2	1/3/2016 20:00:00	8/27/2015 7:50:00
9	2	8/26/2015 19:50:00	7/7/2015 5:07:00
2	3	7/6/2015 17:07:00	6/6/2015 20:57:00
2	12	6/6/2015 8:57:00	3/15/2015 19:48:00
9	12	3/15/2015 7:48:00	10/9/2015 19:30:00
1	7	10/9/2015 7:30:00	8/8/2015 22:56:00
1	14	8/8/2015 10:56:00	3/2/2015 15:36:00
2	4	3/2/2015 3:36:00	9/28/2015 5:12:00
8	3	9/27/2015 17:12:00	11/11/2015 10:22:00
7	13	11/10/2015 22:22:00	4/25/2015 18:48:00
5	12	4/25/2015 6:48:00	9/13/2015 19:37:00
10	2	9/13/2015 7:37:00	9/25/2015 13:22:00
2	12	9/25/2015 1:22:00	1/19/2016 5:24:00
2	11	1/18/2016 17:24:00	10/16/2015 18:51:00
8	7	10/16/2015 6:51:00	11/13/2015 17:07:00
4	7	11/13/2015 5:07:00	7/28/2015 8:54:00
4	1	7/27/2015 20:54:00	2/17/2015 2:09:00
3	13	2/16/2015 14:09:00	12/30/2015 8:12:00
4	10	12/29/2015 20:12:00	2/28/2015 22:28:00
9	13	2/28/2015 10:28:00	11/24/2015 17:41:00
9	7	11/24/2015 5:41:00	8/26/2015 15:51:00
7	13	8/26/2015 3:51:00	12/19/2015 6:09:00
4	13	12/18/2015 18:09:00	6/21/2015 9:46:00
2	11	6/20/2015 21:46:00	10/26/2015 6:11:00
8	6	10/25/2015 18:11:00	5/24/2015 5:29:00
3	2	5/23/2015 17:29:00	11/4/2015 16:01:00
3	15	11/4/2015 4:01:00	10/28/2015 13:27:00
9	3	10/28/2015 1:27:00	5/1/2015 18:37:00
6	9	5/1/2015 6:37:00	7/16/2015 15:31:00
6	1	7/16/2015 3:31:00	3/18/2015 11:25:00
7	10	3/17/2015 23:25:00	8/30/2015 15:43:00

5	13	6/16/2015 4:30:36
10	8	8/10/2015 15:45:23
10	9	3/26/2015 12:50:52
8	20	5/5/2015 11:02:49
6	8	12/6/2015 2:19:05
6	3	3/12/2015 4:32:58
7	4	9/29/2015 8:07:36
9	13	3/14/2015 11:38:43
1	13	6/29/2015 14:59:44
6	1	8/16/2015 1:50:07
5	15	10/10/2015 21:57:02
4	16	8/28/2015 0:15:13
2	16	5/2/2015 20:25:41
1	9	1/29/2016 19:20:14
3	3	5/4/2015 4:30:24
10	13	12/29/2015 14:49:38
5	13	11/24/2015 13:53:40
1	19	9/18/2015 19:46:49
2	5	5/15/2015 7:42:00
4	8	1/3/2016 7:17:40
6	2	5/10/2015 2:35:03
1	3	5/1/2015 15:42:03
3	6	8/8/2015 11:33:04
3	7	6/20/2015 2:47:33
8	9	5/23/2015 23:11:13
3	4	12/16/2015 0:45:41
5	7	5/24/2015 23:30:36
10	14	7/25/2015 18:55:11
6	4	11/11/2015 21:17:45
9	8	8/25/2015 5:22:29
6	5	2/25/2015 10:12:10
1	16	5/25/2015 7:41:34
9	3	1/28/2016 6:21:31
2	12	1/4/2016 3:52:28
1	2	5/13/2015 21:31:27
2	4	7/8/2015 15:08:54
10	3	1/26/2016 3:14:10
10	11	12/9/2015 3:30:58

8	8	8/30/2015 3:43:00	6/19/2015 7:40:00
5	13	6/18/2015 19:40:00	11/9/2015 11:07:00
8	2	11/8/2015 23:07:00	12/16/2015 17:00:00
10	15	12/16/2015 5:00:00	4/6/2015 11:28:00
10	1	4/5/2015 23:28:00	10/23/2015 9:17:00
9	11	10/22/2015 21:17:00	9/16/2015 10:25:00
5	2	9/15/2015 22:25:00	5/1/2015 3:12:00
1	4	4/30/2015 15:12:00	12/22/2015 5:23:00
8	13	12/21/2015 17:23:00	9/16/2015 19:36:00
1	1	9/16/2015 7:36:00	4/7/2015 17:13:00
7	1	4/7/2015 5:13:00	4/8/2015 7:20:00
7	5	4/7/2015 19:20:00	8/17/2015 10:32:00
2	10	8/16/2015 22:32:00	8/21/2015 23:01:00
3	10	8/21/2015 11:01:00	3/11/2015 22:58:00
7	9	3/11/2015 10:58:00	12/7/2015 10:33:00
1	4	12/6/2015 22:33:00	9/6/2015 16:34:00
9	1	9/6/2015 4:34:00	9/22/2015 14:20:00
8	4	9/22/2015 2:20:00	11/6/2015 6:10:00
3	9	11/5/2015 18:10:00	9/23/2015 12:41:00
4	9	9/23/2015 0:41:00	3/10/2015 2:10:00
6	9	3/9/2015 14:10:00	8/13/2015 21:19:00
6	8	8/13/2015 9:19:00	2/15/2015 15:15:00
4	8	2/15/2015 3:15:00	5/20/2015 21:03:00
4	9	5/20/2015 9:03:00	12/7/2015 11:01:00
1	4	12/6/2015 23:01:00	2/3/2016 7:08:00
3	15	2/2/2016 19:08:00	11/26/2015 8:29:00
9	10	11/25/2015 20:29:00	10/21/2015 14:47:00
1	8	10/21/2015 2:47:00	10/21/2015 5:51:00
1	2	10/20/2015 17:51:00	4/2/2015 23:29:00
8	6	4/2/2015 11:29:00	8/7/2015 23:24:00
9	11	8/7/2015 11:24:00	6/23/2015 15:31:00
8	8	6/23/2015 3:31:00	10/11/2015 17:24:00
6	14	10/11/2015 5:24:00	4/7/2015 15:37:00
5	5	4/7/2015 3:37:00	7/11/2015 6:12:00
9	14	7/10/2015 18:12:00	9/4/2015 2:40:00
1	15	9/3/2015 14:40:00	6/18/2015 9:58:00
5	4	6/17/2015 21:58:00	7/18/2015 4:57:00
5	3	7/17/2015 16:57:00	3/19/2015 14:22:00
4	2	3/19/2015 2:22:00	6/24/2015 16:54:00
1	1	6/24/2015 4:54:00	2/15/2015 13:34:00
5	8	2/15/2015 1:34:00	1/0/00 12:00

Special Need							
snID	empID	patientID	activityID	freq	sDate	eDate	date
1	11	9	12	Once daily	2/7/2016	3/4/2016	2/7/2016
2	12	11	5	Three times daily	11/24/2015	12/3/2015	11/24/2015
3	11	15	7	Once daily	4/25/2015	5/19/2015	4/25/2015
4	11	13	6	Twice daily	9/5/2015	9/24/2015	9/5/2015
5	14	12	4	Twice daily	4/26/2015	5/22/2015	4/26/2015
6	11	5	10	Three times daily	11/19/2015	12/1/2015	11/19/2015
7	12	7	9	Once daily	10/3/2015	10/24/2015	10/3/2015
8	13	6	11	Twice daily	6/27/2015	7/5/2015	6/27/2015
9	14	4	15	Twice daily	3/5/2016	3/31/2016	3/5/2016
10	14	10	13	Once daily	4/14/2015	5/11/2015	4/14/2015
11	12	9	12	Once daily	3/8/2016	3/17/2016	3/8/2016
12	12	11	5	Twice daily	12/6/2015	12/31/2015	12/6/2015
13	11	15	7	Twice daily	5/23/2015	6/1/2015	5/23/2015
14	15	13	6	Three times daily	9/27/2015	10/19/2015	9/27/2015
15	12	12	4	Once daily	5/24/2015	6/16/2015	5/24/2015
16	14	5	10	Twice daily	12/3/2015	12/28/2015	12/3/2015
17	14	7	9	Twice daily	10/26/2015	11/16/2015	10/26/2015
18	12	6	11	Once daily	7/7/2015	8/4/2015	7/7/2015
19	15	4	15	Twice daily	4/3/2016	4/15/2016	4/3/2016
20	11	10	13	Once daily	5/14/2015	6/7/2015	5/14/2015

Assessment							
asmtID	empID	patientID	bpSytolic	bpDiastolic	respRate	heartRate	date
5	2	1	177	99	60	151	10/17/2015 13:42:27
5	7	15	119	56	43	91	12/19/2015 16:38:27
3	9	9	199	75	10	111	3/11/2015 14:08:09
8	7	6	125	98	10	149	7/16/2015 0:00:47
7	12	4	120	81	39	46	6/19/2015 8:43:26
1	4	10	137	38	9	119	11/22/2015 15:33:20
6	3	11	166	83	31	36	5/25/2015 7:55:31
4	10	1	107	96	42	63	9/10/2015 6:16:20
7	6	4	105	80	32	148	8/4/2015 7:45:29
5	10	6	180	55	15	121	8/28/2015 14:41:12
2	1	2	140	77	37	179	11/9/2015 21:49:41
10	11	13	194	43	13	23	12/12/2015 15:43:03
1	11	5	110	84	23	107	1/25/2016 21:20:48
4	15	1	109	60	54	106	4/29/2015 12:46:21
6	14	4	190	35	28	104	7/22/2015 6:42:29
8	7	4	166	94	38	95	6/14/2015 3:36:41
2	3	4	165	38	54	58	12/16/2015 13:52:21
1	11	5	178	42	49	44	2/18/2015 6:52:48
8	15	11	116	96	28	83	11/30/2015 0:56:17
10	4	8	126	71	8	145	12/10/2015 10:00:53
7	8	7	150	92	59	150	7/8/2015 11:44:48
7	10	10	121	62	42	165	2/7/2016 12:52:30
2	1	7	158	54	21	103	1/3/2016 17:52:54
7	6	6	168	51	13	176	10/12/2015 21:55:34
10	3	1	172	55	10	158	5/22/2015 21:42:59
10	6	10	138	35	9	118	10/6/2015 1:35:14
10	1	4	177	32	39	129	11/19/2015 12:10:24
7	9	13	152	57	33	118	4/3/2015 10:19:07
9	7	8	197	85	48	112	11/18/2015 16:05:23
1	7	1	130	30	45	104	10/20/2015 2:54:37
3	5	12	184	85	59	111	12/7/2015 5:39:24
3	2	3	197	44	22	96	7/30/2015 9:40:44
2	5	15	135	74	28	103	12/26/2015 8:26:45
1	1	2	160	86	29	41	8/9/2015 23:03:31
8	9	5	123	66	53	43	6/18/2015 3:18:38
9	11	13	189	61	11	124	7/5/2015 1:54:22
8	15	7	105	56	47	75	5/1/2015 6:41:46
5	8	10	122	44	27	35	4/3/2015 0:54:36
1	11	1	159	62	6	104	12/31/2015 20:06:33
9	15	12	200	44	37	115	1/1/2016 7:49:28
9	2	15	113	40	14	145	9/18/2015 5:38:15
4	5	10	171	48	25	42	1/12/2016 9:56:54
7	10	7	172	82	11	104	11/7/2015 9:52:03
7	14	9	153	93	5	104	12/22/2015 2:27:06
9	6	3	197	78	51	167	12/10/2015 18:37:13
2	13	15	105	74	60	51	12/10/2015 3:13:35
3	12	9	105	72	24	62	2/14/2015 21:14:38
10	7	2	199	57	28	94	10/5/2015 23:02:33
9	11	10	137	44	17	103	7/12/2015 4:23:17
9	1	1	102	67	15	125	12/24/2015 15:35:14
10	11	14	116	34	45	90	10/22/2015 17:28:06
6	2	2	173	48	55	76	7/2/2015 7:33:11
7	12	5	162	30	13	90	12/5/2015 17:33:49
1	12	8	169	56	50	85	4/26/2015 6:44:52
8	10	4	185	84	15	172	9/27/2015 19:04:01
9	15	2	168	70	38	103	2/22/2015 22:52:48
7	13	5	193	88	6	168	5/15/2015 17:16:19
1	8	6	180	90	9	147	6/18/2015 22:29:33
2	15	3	174	82	51	168	2/22/2015 15:56:08
8	1	4	176	84	43	174	10/28/2015 13:51:45

Performs		
empID	snID	date
3	6	8/13/2015 6:47:54
9	3	3/29/2015 13:52:58
6	9	4/26/2015 10:38:38
6	9	3/24/2015 7:06:11
2	20	11/24/2015 17:31:04
7	18	3/25/2015 7:58:23
10	8	10/12/2015 18:02:19
4	4	8/26/2015 22:09:31
8	9	3/4/2015 17:45:39
2	17	5/16/2015 20:05:56
10	14	6/6/2015 18:40:18
10	20	10/24/2015 3:35:38
5	1	10/13/2015 17:46:39
9	13	12/17/2015 3:47:44
4	2	2/26/2015 7:42:02
6	20	12/31/2015 10:21:55
1	6	5/17/2015 16:23:29
6	17	10/2/2015 10:03:40
1	15	11/24/2015 11:41:14
9	20	1/5/2016 5:30:09
9	1	5/1/2015 11:15:47
2	4	9/4/2015 19:42:26
10	9	2/26/2015 15:13:41
3	15	3/6/2015 18:07:17
5	6	12/26/2015 3:24:45
6	10	3/6/2015 13:29:32
8	20	2/6/2016 3:40:18
3	18	6/13/2015 13:03:26
2	11	11/30/2015 16:49:15
4	2	5/13/2015 18:17:22
5	19	12/2/2015 20:29:50
7	7	12/4/2015 21:11:18
6	3	6/25/2015 23:54:35
6	13	1/16/2016 18:33:51
7	1	11/5/2015 2:48:37
5	13	3/14/2015 9:47:24
5	13	5/15/2015 1:50:40
8	7	2/9/2016 7:08:21
6	11	4/20/2015 20:44:04

4	8	9/23/2015 23:40:02
7	16	6/4/2015 8:59:54
7	20	4/28/2015 3:00:41
2	16	8/24/2015 15:32:45
8	3	10/24/2015 12:54:44
7	14	6/8/2015 4:00:28
7	15	7/19/2015 6:35:21
5	10	1/9/2016 9:32:54
10	16	2/11/2016 14:50:03
8	13	5/3/2015 1:21:14
1	11	8/10/2015 17:47:40
2	6	5/29/2015 16:51:39
7	15	12/14/2015 5:05:52
4	19	2/1/2016 12:53:59
9	6	6/3/2015 21:56:18
3	13	7/20/2015 8:47:42
5	16	6/15/2015 3:42:38
5	11	9/4/2015 9:21:52
2	10	2/8/2016 4:18:33
6	17	4/4/2015 4:30:01
1	19	8/18/2015 22:41:50
8	7	9/25/2015 14:01:02
3	4	8/8/2015 21:02:18
7	18	9/1/2015 8:49:27
2	4	6/26/2015 11:21:25
2	13	11/13/2015 19:05:13
4	12	8/15/2015 5:47:05
9	18	9/6/2015 10:10:10
3	4	11/25/2015 21:07:50
10	7	12/7/2015 2:32:05
10	16	2/7/2016 20:49:33
2	9	6/17/2015 3:41:04
10	14	9/18/2015 9:54:47
7	17	11/8/2015 12:55:15
2	14	3/7/2015 4:41:45
2	19	6/8/2015 13:36:22
10	19	8/10/2015 6:54:18
7	12	8/8/2015 13:35:43
7	18	3/23/2015 22:39:11
9	10	10/5/2015 23:57:58
9	10	2/28/2015 7:17:53

Works In			
empID	unitID	sDate	eDate
7	4	11/24/2015	2/12/2016
4	5	8/6/2015	10/15/2015
9	5	9/17/2015	12/3/2015
3	4	6/27/2015	9/11/2015
2	5	2/24/2015	5/6/2015
8	1	7/8/2015	9/16/2015
5	2	5/25/2015	7/28/2015
2	3	10/20/2015	12/21/2015
4	1	4/19/2015	6/18/2015
2	1	5/23/2015	7/26/2015
5	2	7/28/2015	10/4/2015
5	3	7/9/2015	9/26/2015
10	5	12/26/2015	3/2/2016
8	5	12/13/2015	2/17/2016
1	2	2/18/2015	5/2/2015
6	5	7/8/2015	9/20/2015
4	4	9/1/2015	11/19/2015
3	5	4/20/2015	6/23/2015
6	1	4/16/2015	6/24/2015
4	4	3/31/2015	6/10/2015
7	4	3/9/2015	5/14/2015
1	4	12/23/2015	2/29/2016
10	5	8/14/2015	10/17/2015
6	2	9/18/2015	12/2/2015
1	4	9/15/2015	11/14/2015
8	4	10/21/2015	12/23/2015
9	5	10/17/2015	1/3/2016
2	1	1/19/2016	4/8/2016
7	4	9/22/2015	12/4/2015
1	1	1/22/2016	3/30/2016

2.4 Sample Queries

Now that we have specified data for our relational database, it is important to specify useful methods to retrieve that data. To illustrate how data is retrieved, we will present sample queries for retrieving data that is useful in very specific scenarios. Each of the queries will be expressed in mathematical expressions for querying relational databases.

2.4.1 Design of Queries

In the following sections, we will describe the three main formal querying languages: relational algebra, tuple relational calculus, and domain relational calculus. Then, we will express our sample queries using the three languages.

2.4.2 Relational Algebra Expressions for Queries

Relational algebra is a set of operations for retrieving tuples from a relational database state. Relational algebra expressions combine the operations to return sets of tuples. Relational algebra expressions describe the process for retrieving tuples from a relational database – in other words, they are procedural, so it is important to order and nest expressions appropriately.

1. List all patients who have been assessed by each of all nurses who are currently hired.

$$\pi_{p.*}\left(\left(\pi_{a.asmtID,\ e.empID}\left(assessment^{a} * employee^{e}\right) \div \pi_{e1.empID}\left(\sigma_{w.eDate\ !=\ NULL}\left(employee^{e1} * workIn^{w}\right)\right)\right) * assessment^{a1} * patient^{p}\right)$$

2. List all nurses who have been assigned to least 2 patients in the same time period.

$$\pi_{e.*}\left(\left(\sigma_{\substack{a1.patientID\ !=\ a2.patientID\ \wedge \\ a1.sDate \geq a2.sDate\ \wedge \\ a1.sDate \leq a2.eDate}}\left(assigned^{a1} \times assigned^{a2}\right)\right) * employee^{e}\right)$$

3. List the medications that all currently admitted patients have in common.

$$\left(\pi_{p.patientID,\ m.medID}\left(\sigma_{\substack{rx.sDate \leq today\ \wedge \\ rx.eDate \geq today}}\left(prescription^{rx} * medication^{m} * patient^{p}\right)\right) \div \pi_{p.patientID}\left(\sigma_{pa.dischargedDate\ =\ NULL}\left(patient^{p} * patientAdmitted^{pa}\right)\right)\right) * medication$$

4. List all patients who have been admitted to the hospital exactly once.

$$\left(\pi_{pa.patientID}\left(patientAdmitted\ a\right) - \pi_{pa1.patientID}\left(\sigma_{pa1.patientID = pa2.patientID \wedge pa1.admittedDate \neq pa2.admittedDate}\left(patientAdmitted\ pa1 \times patientAdmitted\ pa2\right)\right)\right) * patient\ p$$

5. List all patients who have received special need orders for all of the same activities as John Doe ordered for Chelsea Doe.

$$\left(\pi_{p.patientID,\ a.activityID}\left(patient\ p * specialNeed\ s * activity\ a\right) \div \pi_{a2.activityID}\left(\sigma_{p2.fName = "Chelsea" \wedge p2.lName = "Doe" \wedge e.fName = "John" \wedge e.lName = "Doe"}\left(patient\ p * specialNeed\ s2 * activity\ a2 * employee\ e\right)\right)\right) * patient$$

6. List all of the patients who have been assessed today by at least one of the nurses who assessed John Doe on 1/27/15.

$$\pi_{p.*}\left(\sigma_{a.date = today}\left(\pi_{e.*}\left(\sigma_{p1.fName = "John" \wedge p1.lName = "Doe" \wedge a1.date = "1/27/16}\left(employee\ e * assessment\ a1 * patient\ p1\right)\right) * assessment\ a\right) * patient\ p\right)$$

7. List the nurses who most recently administered one of Chelsea Doe's prescriptions.

$$\pi_{e.*}\left(\pi_{a.*}\left(\sigma_{p.fName = "Chelsea" \wedge p.lName = "Doe"}\left(patient\ p * \left(prescription\ rx \bowtie administers\ a\right)\right)\right) - \pi_{a1.*}\left(\sigma_{p1.fName = "Chelsea" \wedge p1.lName = "Doe" \wedge p2.fName = "Chelsea" \wedge p2.lName = "Doe" \wedge a1.date < a2.date}\left(\left(patient\ p1 * \left(prescription\ rx1 \bowtie_{rx1.rxID = a1.rxID} administers\ a1\right)\right) \times \left(patient\ p2 * \left(prescription\ rx2 \bowtie_{rx2.rxID = a2.rxID} administers\ a2\right)\right)\right)\right) * employee\ e\right)$$

8. List nurses who have performed each of all activities in the database.

$$\left(\pi_{p.empID,\ a.activityID}\left(\left(specialNeed\ s \bowtie_{s.snID = p.snID} performs\ p\right) * activity\ a\right) \div \pi_{a2.activityID}\left(activity\ a2\right)\right) * employee\ e$$

9. List all discharged patients who stayed in the same room as John Doe.

$$\pi_{p.*}\left(\left(\pi_{pl1.roomID}\left(\sigma_{\substack{p1.fName = "John" \wedge \\ p1.lName = "Doe" \wedge}}\left(placed^{pl1} * patient^{p1}\right)\right) * placed^{pl2}\right) * \left(patient^{p} - \pi_{p2.*}\left(\sigma_{pa.dischargedDate = NULL}\left(patient^{p2} * patientAdmitted^{pa}\right)\right)\right)\right)$$

10. List all currently-admitted patients who don't have any currently active prescriptions.

$$admittedPatients \leftarrow \pi_{p.*}\left(\sigma_{pa.dischargedDate = NULL}\left(patient^{p} * patientAdmitted^{pa}\right)\right)$$

$$admittedPatients - \pi_{p.*}\left(\sigma_{rx.sDate \leq today \wedge rx.eDate \geq today}\left(admittedPatients * prescription^{rx}\right)\right)$$

2.4.3 Tuple Relational Calculus Expressions for Queries

Relational calculus is a querying language that uses declarative, nonprocedural expressions; relational calculus expressions describe the set of tuples that will be retrieved, without specifying the order of operations needed to retrieve it. Relational calculus uses free variables (the variables that describe what the query will retrieve), bound variables (variables bounded by the existential or universal quantifiers), and logical expressions with truth value. There are two variations of relational calculus – tuple and domain. In tuple relational calculus, each variable represents a tuple – a list of values that satisfy the attribute domains of a relational schema.

1. List all patients who have been assessed by each of all nurses who are currently hired.

```
{ p | patient(p) ^  (∀e) (  Employee(e) ^ ( ∃w)(works_in(w) ^ w.eDate != NULL ^ w.empID = e.empID)
        → ∃a ( Assessment(a) ^ a.empID = e. empID ^ a.patientID = p.patientID) )
}
```

2. List all nurses who have been assigned to least 2 patients in the same time period.

```
{ e | Employee(e) ^ (∃a1) (∃a2) ( Assigned(a1) ^ Assigned(a2)
        ^ a1.empID = e.empID ^ a2.empID = e.empID
        ^ a1.patientID != a2.patientID
        ^ a1.sDate >= a2.sDate
        ^ a1.sDate <=a2.eDate)
}
```

3. List the medications that all currently admitted patients have in common.

```
{ m | Medication(m) ^ (∀p) ( Patient(p) ^ ∃pa ( PatientAdmitted(pa) ^ pa.patient = p.patientID
        ^ pa.dischargedDate = NULL )
        → ∃rx ( Prescription(rx)
                ^ rx.sDate ≤ today ^ rx.eDate ≥ today ^ rx.medID = m.medID
                ^ rx.patientID = p.patientID ))
}
```

4. List all patients who have been admitted to the hospital exactly once.

```
{ p | Patient(p) ^ (∃pa) ( PatientAdmitted(pa) ^ pa.patientID = p.patientID)
        ^ ¬∃pa2( PatientAdmitted(pa2)
        ^ pa.patientID = pa2.patientID ^ pa.admittedDate != pa2.admittedDate)
}
```

5. List all patients who have received special need orders for all of the same activities as John Doe ordered for Chelsea Doe.

{ p | Patient(p) ^ (∀p2) (∀s) (∀e) (Patient(p2) ^ SpecialNeed(s) ^ Employee(e)
^ s.patientID = p2.patientID
^ s.empID = e.empID ^ p2.fName = “Chelsea” ^ p2.lName = “Doe”
^ e.fName = “John” ^ e.lName= “Doe”
→ ∃s2 (SpecialNeed(s2) ^ s2.activityID = s.activityID ^ p.patientID = s2.patientID))
}

6. List all of the patients who have been assessed today by at least one of the nurses who assessed John Doe on 1/27/15.

{ p | Patient(p) ^ (∃e) (∃a1) (∃p1) (Employee(e) ^ Assessment(a1) ^ Patient(p1) ^ a1.empID = e.empID
^ a1.patientID = p1.pateintID ^ p1.fName = “John” ^ p1.lName = “Doe”
^ a1.date = “1/27/16” ^ ∃a2 (Assessment(a2)
^ a2.empID = e.empID ^ a2.patientID = p.patientID
^ a2.date = today))
}

7. List the nurses who most recently administered one of Chelsea Doe’s prescriptions.

{ e | Employee(e) ^ (∃p) (∃rx) (∃a) (Patient(p) ^ Prescription(rx) ^ Administers(a)
^ rx.patientID = p.patientID ^ a.rxID = rx.rxID
^ p.fName = “Chelsea” ^ p.lName = “Doe” ^ a.empID = e.empID
^ ¬ ∃p2 ∃rx2 ∃a2(Patient(p2) ^ Prescription(rx2) ^ Administers(a2)
^ rx2.patientID = p2.patientID ^ a2.rxID = rx2.rxID
^ p.fName = “Chelsae” ^ p.lName = “Doe” ^ a2.Date > a.Date))
}

8. List nurses who have performed each of all activities in the database.

{ e | Employee(e) ^ (∀a) (Activity(a)
→ ∃s ∃p (SpecialNeed(s) ^ Performs(p)
^ s.activityID = a.activity ID
^ p.snID = s.snID
^ p.empID = e.empID))
}

9. List all discharged patients who stayed in the same room as John Doe.

{ p | Patient(p) ^ (∃p1) (∃pl1) (Patient(p1) ^ Placed(pl1) ^ pl1.patientID = p1.patientID
^ p1.fName = “John” ^ p1.lName= “Doe” ^ ∃pl2 (Placed(pl2)
^ pl2.patientID = p.patientID ^ pl2.roomID = pl1.roomID))
^ ¬ ∃pa (PatientAdmitted(pa) ^ pa.patientID = p.patientID
^ pa.dischargedDate = NULL)
}

10. List all currently admitted patients who don't have any currently active prescriptions.

{ p | Patient(p) ^ (∃pa) (PatientAdmitted(pa) ^ pa.dischargedDate = NULL
^ pa.patientID = p.patientID) ^ ¬ ∃rx (Prescription(rx)
^ rx.sDate ≤ today ^ rx.eDate ≥ today
^ rx.patientID = p.patientID)
}

2.4.4 Domain Relational Calculus Expressions for Queries

Domain relational calculus is a variation of relational calculus. In domain relational calculus, each variable represents a single value within a tuple, instead of a tuple itself.

1. List all patients who have been assessed by each of all nurses who are currently hired.

{ < p, f, l> | Patient(p, _, f, _, l, _, _, _, _, _, _, _, _, _)
^ (∀e) (Employee(e, _, _, _, _, _, _, _, _, _, _, _) ^ works(e, _, _, != NULL)
→ (∃a) (Assessment(a, e, p, _, _, _, _)))
}

2. List all nurses who have been assigned to least 2 patients in the same time period.

{ < e, f, l > | Employee(e, _, fName, _, lName, _, _, _, _, _, _, _)
^ (∃p1) (∃p2) (∃sD) (Assigned(e, p1, sD, _) ^ Assigned(e, p2, <= sD, >= sD)
^ p1 != p2)
}

3. List the medications that all currently admitted patients have in common.

{ < m, n, p > | Medication(m, n, p)
^ (∀p) (Patient (p, _, _, _, _, _, _, _, _, _, _, _ , _, _, _) ^ PatientAdmitted(p, _, _, NULL)
→ (∃rx) (Prescription(rx, _, p, m, _, _, <= today, >= today, _)))
}

4. List all patients who have been admitted to the hospital exactly once.

{ < p, f, l> | Patient(p, _, f, _, l, _, _, _, _, _, _, _, _, _)
^ (∃d) (PatientAdmitted(p, d, _) ^ ¬(∃d2) (PatientAdmitted(p, d2, _) ^ d != d2))
}

5. List all patients who have received special need orders for all of the same activities as John Doe ordered for Chelsea Doe.

{ <p, f, l> | Patient(p, _, f, _, l, _, _, _, _, _, _, _, _, _)
^ (∀p2) (∀e)(∀a) (Patient(p2, _, “Chelsea”, _, “Doe”, _, _, _, _, _, _, _, _, _)
^ Employee(e, _, “John”, _, “Doe”, _, _, _, _, _, _, _, _)
^ SpecialNeed(_, e, p2, a, _, _, _, _)
→ SpecialNeed(_, _, p, a, _, _, _, _))
}

6. List all of the patients who have been assessed today by at least one of the nurses who assessed John Doe on 1/27/15.

{ <p, f, l> | Patient(p, _, f, _, l, _, _, _, _, _, _, _, _, _)
^ (∃e) (∃p2) (Patient(p2, _, “John”, _, “Doe”, _, _, _, _, _, _, _, _, “1/27/16”)
^ Assessment(_, e, p2, _, _, _, _, _)
^ Assessment(_, e, p, _, _, _, _, today))
}

7. List the nurses who most recently administered one of Chelsea Doe’s prescriptions.

{ <e, f, l> | Employee(e, _, f, _, l, _, _, _, _, _, _, _)
^ (∃p) (∃rx) (∃d) (Patient (p, _ , “Chelsea”, _, “Doe”, _, _, _, _, _, _, _, _, _)
^ Prescription (rx, _, p, _, _, _, _, _,)
^ Administers (e, rx, d)
^ ¬ (∃rx2) (∃d2) (Prescription (rx2, _, p, _, _, _, _, _) ^ Administers(_, rx2, d2)
^ d2 > d))
}

8. List nurses who have performed each of all activities in the database.

{ <e, f, l> | Employee(e, _, f, _, l, _, _, _, _, _, _, _)
^ (∀a) (Activity(a, _, _)
→ ∃s (SpecialNeed(s, _, _, a, _, _, _, _) ^ Performs(e, s, _)))
}

9. List all discharged patients who stayed in the same room as John Doe.

{ <p, f, l> | Patient(p, _, f, _, l, _, _, _, _, _, _, _, _, _)
^ ¬ PatientAdmitted(p, _, NULL)
^ (∃p2) (∃r) (Patient(p2, _, “John”, _, “Doe”, _, _, _, _, _, _, _, _, _)
^ Placed (p2, r, _, _)
^ Placed(p, r, _, _))
}

10. List all currently-admitted patients who don't have any currently active prescriptions.

{ <p, f, l> | Patient(p, _, f, _, l, _, _, _, _, _, _, _, _, _)
^ PatientAdmitted(p, _, NULL)
^ ¬ Prescription(_, _, p, _, _, _, <= today, >= today)
}

3. Oracle Database Management System

3.1 Normalization of Relations

Before we implement our logical database as a physical database, it is important to analyze the quality of the relational database schema design. In this section, we will define a formal method called normalization for measuring whether a relational database schema is well designed, and we will explore some of the problems that can occur when applying operations to a poorly normalized database design once it is implemented. Then, we will analyze the relation schemas in the Charity General Hospital logical design

3.1.1 Normalization and Normal Forms

Before we analyze the quality of the relation schemas in the Charity General Hospital database, we should define the concepts of normalization, as well as how to measure normalization using normal forms.

Description of Normalization and Normal Forms

A relational database schema can be poorly designed so that tuples contain redundant data. This is problematic because if the value for a redundant attribute is changed in one tuple, it may have to be changed in several others for the data to remain coherent – this is known as a modification anomaly. Normalization is the process of decomposing, or "breaking apart" relation schemas so that this redundancy is removed and modification anomalies do not occur.
In order to test how well each relation schemas is normalized, we will use a formal method for measuring normalization called *normal form tests*. There are four main normal forms that a relation schema can satisfy or belong to: first, second, third, and Boyce-Codd. The normal forms can be used to rank relation schemas – the higher the normal form, the more normalized the schema is. In a well-designed relational database, all schemas should satisfy at least third normal form.

The conditions that must be satisfied for each normal form will now be explained in detail.

First Normal Form
In order for first normal form (1NF) to be satisfied, all attribute values in a relation schema must single and atomic. In other words, 1NF does not allow relation tuples that contain nested tuples of values. There are several methods to decompose a relation schema that is not (1NF). The multi-value attribute can be made into a separate relation that contains the original relation's primary key as a foreign key attribute. This method is used to map multi-value attributes in the ER-model to relations in the previous chapter. Another method can be used if the multi-value attribute contains a specific number of values for each tuple; in this case, a new single-value attribute is added to the relation schema for each value in the multi-value attribute. Finally, the multi-value attribute replaced with a new single-value attribute,

and each of the multiple values will be stored in a separate, “duplicate” tuple. For example, if the attribute value contains three nested sub values, there will be three tuples that each contain one of the multiple values, and are otherwise identical. The final solution creates redundant data, so it should be avoided.

Second Normal Form

Second normal form (2NF) evaluates relational schema attributes based on *functional dependency*. A set of attributes Y is functionally dependent on another set of values X if the set of values for X map to only one set of values for Y. In other words, the values of X can be used to determine the values of Y. An example of a functional dependency is the primary key of a tuple, because each primary key value maps to only one tuple. Functional dependency is notated as:

$$X \rightarrow Y$$

Y functionally depends on X.

In order for a relation schema to satisfy 2NF, it must satisfy 1NF. Also, all attributes that are not part of the primary key must *fully functionally depend* on the primary key. Attributes fully functionally depend on the primary key if the functional dependency no longer holds once one of the attributes for the primary key is removed; in other words, nonprime attributes (a term for attributes that are not part of the primary key) must functionally depend on the entire primary key, not just a part of it.
All relation schemas that have a single-attribute primary key automatically past the 2NF test. If a relational schema fails the test, it can be normalized by being broken into smaller relation schemas whose primary keys are subsets of the original primary key.

Third Normal Form

In order for a relation schema to satisfy Third Normal Form (3NF), it must satisfy 2NF and 1NF. Also, there cannot be any non-prime attributes that functionally depend on other non-prime attributes. In other words, no non-prime attributes can *transitively* depend on the primary key.
If a relation schema fails the 3NF test, then it can be normalized by being decomposed into relations where the left side of a functional dependency is always a primary key attribute (or a superkey which contains the primary key).

Boyce-Codd Normal Form

Boyce-Codd Normal Form (BCNF) is very similar to 3NF, but is stricter. In order for a relation schema to satisfy BCNF, all previous normal forms must be satisfied, and the left side of any functional dependency must be a primary key (or superkey) of the relation schema. BCNF is stricter than 3NF because it does not allow any prime attributes (members of primary or candidate keys) to depend on non-prime attributes.

If a relation schema fails the BNF test, the schema can be decomposed into relations where the non-prime attributes at the left side of any functional dependencies become prime attributes of the new relation schemas.

Anomalies that Result from Poor Normalization

Unless 3NF or BCNF at minimum are satisfied, relations will contain redundant data that cause anomalies to occur when data is modified. These anomalies can be divided into three classes: insertion, modification, and deletion.

Insertion Anomalies

Redundant data that violates 2NF, 3NF, or BCNF tests can be created by storing the natural join of two relations as one relation. Several anomalies or incoherencies can occur when trying to insert new tuples for the "joined" relation schema.

First, to insert two tuples that represent two relations that are both joined to the same relation, the attribute values for the joined relation must be *exactly* the same for both tuples in order for the data to be coherent.

Second, to insert a tuple representing a relation that is not joined to any other relations, the attribute values for the other relation schema must all be set to NULL. This creates problems because NULL values have many possible interpretations, including "absent," "not applicable," and "unknown." Also, if any of the attributes that are set to NULL help compose the primary key of the joined relation, then the entity integrity constraint will be violated (see Phase 2).

Update Anomalies

If a set of tuples can represent one single relation that is joined to several other relations, then the attribute values representing the single relation will appear in all of the tuples. If any of those attribute values at are changed in one tuple, they must be changed in *all* of the tuples in order for the data to remain coherent.

Deletion Anomalies

If a set of tuples can represent one single relation that is joined to several other relations, and all of the tuples are removed, then any record of the single relation will be completely removed from the database. In other words, it will be impossible to join the single relation to other relations in the future.

3.1.2 Normal Forms for This Database

We will now analyze each of the relation schemas in the database to determine if they at least satisfy 3NF. Then, we will analyze a relation schema that results from joining two relation schema and discuss anomalies that can occur as a result.

Employee

Functional Dependencies:
FD1. Trivial; {**empID**} → {SSN, fName, mName, …, licNo}
FD2. {*SSN*} → {**empID**, fName, mName, …, licNo}

Candidate Keys:
empID

Normal Form:
1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because the primary key only has one attribute.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes.
BNF is satisfied because the left side of all functional dependencies is a candidate key (SSN is a candidate key).

BNF is satisfied, so there should be no inherent modification anomalies.

Prescription

Functional Dependencies:
FD1. Trivial; {**rxId**} → {empID, …, rxDate}

Candidate Keys:
rxID

Normal Form:
1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because the primary key only has one attribute.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes.
BNF is satisfied because the left side of all functional dependencies is a candidate key.

BNF is satisfied, so there should be no inherent modification anomalies.

Medication

Functional Dependencies:

FD1. Trivial; {**medID**} → {medName, medPurpose}

FD2. Trivial; {*medName*} → {**medID**, medPurpose}

Candidate Keys:

medID, medName

Normal Form:

1NF is satisfied because all attributes have atomic domains.

2NF is satisfied because the primary key only has one attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BNF is satisfied because the left side of all functional dependencies is a candidate key.

BNF is satisfied, so there should be no inherent modification anomalies.

Room

Functional Dependencies:

FD1. Trivial; {**roomID**} → {unitID, roomNo}

FD2. {unitID, roomNo} → {roomID}

Candidate Keys:

roomID, {unitID, roomNo}

Normal Form:

1NF is satisfied because all attributes have atomic domains.

2NF is satisfied because the primary key only has one attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BNF is satisfied because the left side of all functional dependencies is a candidate key ({untID, roomNo} is a candidate key).

BNF is satisfied, so there should be no inherent modification anomalies.

Unit

Functional Dependencies:

FD1. Trivial; {**unitID**} → {unitName, unitAbbrv}

FD2. {unitName} → {**unitID**, unitAbbrv}

FD3. {unitAbbrv} → {**unitID**, unitName}

Candidate Keys:

unitID, unitName, unitAbbrv

Normal Form:

1NF is satisfied because all attributes have atomic domains.

2NF is satisfied because the primary key only has one attribute.

3NF is satisfied because there are no non-prime attributes.

BNF is satisfied because the left side of all functional dependencies is a candidate key.

BNF is satisfied, so there should be no inherent modification anomalies.

Patient

Functional Dependencies:

FD1. Trivial; {**patientID**} → {SSN, fName, mName, …, licNo}

FD2. {*SSN*} → {**patientID**, fName, mName, …, licNo}

Candidate Keys:

patientID, SSN

Normal Form:

1NF is satisfied because all attributes have atomic domains.

2NF is satisfied because the primary key only has one attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BNF is satisfied because the left side of all functional dependencies is a candidate key (SSN is a candidate key).

BNF is satisfied, so there should be no inherent modification anomalies.

Patient Admitted

Functional Dependencies:

FD1. Trivial; {patientID, admittedDate} → {dischargedDate}

Candidate Keys:

{patientID, admittedDate}

Normal Form:

1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because no attributes functionally depend on only part of the primary key.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes.
BNF is satisfied because the left side of all functional dependencies is a candidate key.

BNF is satisfied, so there should be no inherent modification anomalies.

Placed

Functional Dependencies:

FD1. Trivial; {patientID, roomID, sDate} → {eDate}

Candidate Keys:

{patientID, roomID, sDate}

Normal Form:

1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because no attributes functionally depend on only part of the primary key.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes.
BNF is satisfied because the left side of all functional dependencies is a candidate key.

BNF is satisfied, so there should be no inherent modification anomalies.

Assessment

Functional Dependencies:

FD1. Trivial; {**asmtID**} → {empID,…,date}

FD2. {patientID, date} → {empID, …}

FD3. {empID, date} → {patientID, …}

Candidate Keys:

asmtID, {patientID, date}, {empID, date}

*the non-primary candidate keys are unique because date is of type “timestamp”

Normal Form:

1NF is satisfied because all attributes have atomic domains.

2NF is satisfied because the primary key only has one attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BNF is satisfied because the left side of all functional dependencies is a candidate key.

BNF is satisfied, so there should be no inherent modification anomalies.

Special Need

Functional Dependencies:

FD1. Trivial; {**snID**} → {empID,…,date}

Candidate Keys:

snID

Normal Form:

1NF is satisfied because all attributes have atomic domains.

2NF is satisfied because the primary key only has one attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BNF is satisfied because the left side of all functional dependencies is a candidate key.

BNF is satisfied, so there should be no inherent modification anomalies.

Administers

Functional Dependencies:

FD1. {rxID, date} → {empID}

FD2. {empID, date} → {rxID}

Candidate Keys:

{rxID, date}, {empID, date}

Normal Form:

1NF is satisfied because all attributes have atomic domains.

2NF is satisfied because the primary key only has one attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BNF is satisfied because the left side of all functional dependencies is a candidate key.

BNF is satisfied, so there should be no inherent modification anomalies.

Assigned To

Functional Dependencies:

FD1. Trivial; {empID, patientID, sDate} → {eDate}

Candidate Keys:

{empID, patientID, sDate}

Normal Form:

1NF is satisfied because all attributes have atomic domains.

2NF is satisfied because the primary key only has one attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BNF is satisfied because the left side of all functional dependencies is a candidate key.

BNF is satisfied, so there should be no inherent modification anomalies.

Performs

Functional Dependencies:

FD1. {snID, date} → {empID}

FD2. {empID, date} → {snID}

Candidate Keys:

{snID, date}, {empID, date}

Normal Form:

1NF is satisfied because all attributes have atomic domains.

2NF is satisfied because no attributes depend on only part of the primary key.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BNF is satisfied because the left side of all functional dependencies is a candidate key.

BNF is satisfied, so there should be no inherent modification anomalies.

Works In

Functional Dependencies:

FD1. {empID, unitID, sDate} → {eDate}

Candidate Keys:

{empID, unitID, sDate}

Normal Form:

1NF is satisfied because all attributes have atomic domains.

2NF is satisfied because the primary key only has one attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BNF is satisfied because the left side of all functional dependencies is a candidate key.

BNF is satisfied, so there should be no inherent modification anomalies.

Example of Poorly Normalized Relation: Prescription-Medication

Prescription-Medication is a relation schema created by natural joining the Prescription and Medication relations. We will show its functional dependencies, explain why it does not satisfy all of the normal form tests, and illustrate some of the anomalies that can occur as a result.

Relation:
prescriptionmedication(**rxID**, empID, patientID, dosage, freq, sDate, eDate, rxDate, medName, medPurpose)

Functional Dependencies:
FD1. {**rxID**} → {empID, …, medName, medPurpose}
FD2. {medName} → {medPurpose}

Candidate Keys:
rxID

Normal Form:
1NF and 2NF are satisfied, but 3NF is not. This is because *medPurpose* functionally depends on a nonprime attribute *medName*, and thus transitively depends on the primary key.

Possible Anomalies:
If the same medication is prescribed multiple times, both the *medName* and *medPurpose* fields must be exactly the same for all tuples so that the data remains consistent. If the *medName* is modified in one of the tuples with an UPDATE operation, it must be modified in all the tuples for which the medication is prescribed. In order to add a medication to the database that hasn't been prescribed, the prescription fields all have to be "NULL." If all prescriptions for a medication are removed from the database, then any record of the medication will be removed as well.

3.2 SQL*PLUS: Main Purpose and Functionality

Now that the quality of the relational design has been proven, we will describe the physical implementation process. The physical database was implemented and loaded with sample data using SQL * Plus. SQL * Plus is a command-line user interface for interacting with the Oracle DBMS. The main purpose of SQL * Plus is to allow database administrators to quickly define and easily maintain a database. It allows users to enter SQL commands to define and manage schema objects, manipulate and query existing data, and control the formatting of output. It also allows users to create and run scripts that execute multiple of the above commands at once. Finally, SQL * Plus allows users to create and run PL/SQL scripts; PL/SQL is Oracle's procedural extension of SQL, combining SQL statements with flow control structures like conditions and loops. PL/SQL programs can be saved as stored procedures and set to automatically run using triggers.

3.3 Schema Objects for Oracle DBMS

Tables

Tables are a basic unit of data storage in an Oracle database. The data is stored in rows which contain the attributes of the relational schema. In a table, columns have column names (such as emp_id, first_name, and start_date), a specified datatype (such as NUMBER, VARCHAR2, or TIMESTAMP), and a width. Once the data are inserted into the tables, it can then be updated, deleted, or queried using SQL language. Tables may also contain virtual columns which derive values on demand through user specified functions.

Syntax:

create table **<tablename>** (
    <column-definition-1>,
    …
    <column-definition-n>,
        <table constraints>
)

<column-definition>:= **<column-name>** **<column-datatype>** **<column-constraints>**

<table constraints>:=
constraint **<constraint-name>** primary key (**<column-name>**),
foreign key (**<column-name>**) references **<table-name>** (**<column-name>**),
unique (**<column-name>**),
check **<boolean-expression>**

Examples in this Implementation:

- arjt_activity
- arjt_administers
- arjt_assignment_old
- arjt_assessment
- arjt_assigned_to
- arjt_employee
- arjt_medication
- arjt_patient_admitted
- arjt_patient
- arjt_performs
- arjt_placed
- arjt_prescription
- arjt_room
- arjt_special_need
- arjt_unit
- arjt_works_in

Views

Views are the results of a query stored as virtual tables; they do not store data, but a SELECT statement that generates a specific representation of data. This way, a DBA can control which data is available, how it is presented, and how it is formatted before it is queried by other users. Often, front-end applications query from views instead of tables to save time and simplify queries. Many of the operations that apply to tables can be applied to views, including SELECT, CREATE, INSERT, UPDATE, and DELETE; these operations modify information in the base tables from which the views are derived. When information in the base tables changed, views are dynamically recreated.

Syntax:
create or replace view **<view_name>** AS **<select statement/query>**

Procedures

Procedures are stored blocks of PL/SQL code that can be run via the command line or in scripts. Because procedures are stored, they are reusable; thus, they are often used to perform repetitive manipulations on data. Procedures take advantage of the flow control structures provided by PL/SQL, so they can perform more complex operations than pure SQL scripts. They can be invoked in triggers, so that the stored procedure runs each time a specific event occurs.

Syntax:
create or replace procedure **<procedure name>** begin **<PL/SQL statements>** end

Note: Procedures will be implemented during Phase 4 of the process.

Triggers

Triggers are stored blocks of PL/SQL code that automatically run each time a specific event occurs (like an INSERT operation).

Syntax:
create or replace trigger **<triggername>** after **<event name>** on **<table name>**
begin **<PL/SQL statements>** end

Packages

Packages are groupings of PL/SQL objects and procedures that provide an interface to more complicated SQL * Plus functionality. They abstract and encapsulate data and functions similarly to classes in object-oriented programming. Packages contain a spec block that defines the public interface to the package, as well as a "body" block which fully defines the hidden, abstracted code within procedures. Packages allow complex procedures to be easily reused and hidden from front-end application developers.

Syntax:
create or replace package **<package_name**> as **<object and procedure declarations>** end
create or replace package body **<package_name>** as <**object and procedure definitions>** end

Sequence Generators

Sequence generators use a mathematical function to produce a sequence of unique values. Each time the sequence generator is requested, it responds with the next number in the sequence. Sequence generators are often used to generate unique values for primary key attributes, and ensure that unique primary key values are used for new tuples being inserted by multiple users at the same time. Sequence generators have a caching option which allows the generator to pre-calculate and store the next *n* numbers in the sequence in memory.

Syntax:

create sequence **<sequence name>**
minvalue **<minimum value>**
maxvalue **<maximum value>**
start with **<starting number>**
increment by **<increment size>**
cache **<cache_size>**

Examples in this implementation:

- arjt_prescription_id_sequence
- arjt_room_id_sequence
- arjt_unit_id_sequence
- arjt_assessment_id_sequence
- arjt_specialneed_id_sequence
- arjt_activity_id_sequence

Indexes

Creating indexes provide faster access paths to specified table columns which speed up queries. Columns may be used in multiple indexes as long as each index contains a unique set of columns. Oracle automatically creates indexes for primary keys. Indexes are logically and physically independent as they may be created and dropped at anytime without affecting the table data or other indexes, although previously indexed data can be slower. Oracle provides the following indexing schemes which correspond to speed improvements.

- B-tree indexes
- B-tree cluster indexes
- Hash cluster indexes
- Reverse key indexes
- Bitmap indexes
- Bitmap join indexes

3.4 List Relations With SQL Commands

Employee:

```
WINTER342 SQL> desc arjt_employee
 Name                      Null?    Type
 ------------------------- -------- ----------------
 EMP_ID                    NOT NULL NUMBER(5)
 SSN                       NOT NULL NUMBER(9)
 FIRST_NAME                NOT NULL VARCHAR2(30)
 MID_NAME                           VARCHAR2(30)
 LAST_NAME                 NOT NULL VARCHAR2(30)
 STREET                    NOT NULL VARCHAR2(50)
 CITY                      NOT NULL VARCHAR2(30)
 STATE                     NOT NULL VARCHAR2(30)
 ZIP                       NOT NULL NUMBER(5)
 PHONE                     NOT NULL NUMBER(10)
 LIC_TYPE                  NOT NULL VARCHAR2(2)
 LIC_NO                    NOT NULL NUMBER(6)

WINTER342 SQL> spool off
```

```
WINTER342 SQL> select * from arjt_employee;

 EMP_ID        SSN FIRST_NAME  MID_NAME    LAST_NAME    STREET                      CITY            STATE           ZIP      PHONE       LI    LIC_NO
------- ---------- ---------- ---------- ------------ --------------------------- --------------- --------------- -------- ----------- ----- ---------
      1  881076941 Albert     Terry      Perry        822 Rowland Parkway         Korkino         California      87689    3215391654  RN    603812
      2  531711882 Arthur     Willie     Moore        8036 Rigney Point           Cambas          Texas           92127    2929343205  RN    365620
      3  118454889 Ruby       Doris      Reid         71112 Grasskamp Junction    Pattani         Colorado        52743    4655925227  RN    261701
      4  171400011 Deborah    Louis      Gordon       917 Golf Course Trail       Makato          Mississippi     26234    9451883271  RN    396551
      5  792092438 Jose       Louis      Day          31739 Fremont Pass          Washington      North Carolina  13524    9060612037  RN    342062
      6  485255265 Stephanie  Anne       Johnson      16849 Ramsey Alley          Alukama         Oregon          84370    9843948439  RN    328804
      7  777181715 William    Bentley    Price        1827 Michigan Point         Barbacena       Washington      89349    5106048310  RN    980387
      8  158853851 Ray        William    Wells        9527 Cody Terrace           Princeton       Colorado        51324    8773206927  RN    916003
      9  763896972 Joyce      Maria      Price        293 Merchant Terrace        Santo Amaro     Florida         27256    2954554242  RN    809919
     10  959716420 Mark       Allen      Jefferson    5030 Hagan Crossing         Nashville       Texas           92723    3102260136  RN    543560
     11  669336025 Samuel     Mich       Taylor       7730 Scoville Drive         Francisco Villa California      55310    1230115500  MD    240773
     12  916527281 David      Brian      Madson       8508 Starling Crossing      Fresno          California      63509    6807569285  MD    669696
     13  321693401 Jose       Walter     Hall         296 Upham Plaza             Grand Rapids    Texas           77201    7274041819  MD    901056
     14  321083187 Harold     Lawrence   Roberts      5041 Michigan Way           Antigonish      Colorado        38103    7396402185  MD    286169
     15  907309890 Ruth       Stephanie  Hamilton     6283 Magdeline Place        Brooklyn        Pennsylvania    88929    9165982423  MD    926923
15 rows selected.
WINTER342 SQL> spool off
```

Prescription:

```
WINTER342 SQL> desc arjt_prescription
 Name                      Null?    Type
 ------------------------- -------- ----------------
 RX_ID                     NOT NULL NUMBER(5)
 EMP_ID                    NOT NULL NUMBER(5)
 PAT_ID                    NOT NULL NUMBER(5)
 MED_ID                    NOT NULL NUMBER(5)
 DOS                       NOT NULL VARCHAR2(50)
 FREQ                      NOT NULL VARCHAR2(100)
 START_DATE                NOT NULL DATE
 END_DATE                  NOT NULL DATE
 DAT                       NOT NULL DATE

WINTER342 SQL> spool off
```

```
WINTER342 SQL> select * from arjt_prescription;

RX_ID EMP_ID PAT_ID MED_ID DOS      FREQ              START_DATE  END_DATE    DAT
----- ------ ------ ------ -------- ----------------- ----------- ----------- -----------
    1     12      1      3 113 mg   Three times daily 22-AUG-14   27-SEP-14   21-AUG-14
    2     13      3      4 109 mg   Once daily        08-JUL-15   14-AUG-15   07-JUL-15
    3     13     14      4 83 mg    Three times daily 21-AUG-14   30-SEP-14   19-AUG-14
    4     13     14     10 177 mg   Three times daily 25-SEP-14   01-NOV-14   23-SEP-14
    5     13      1      4 168 mg   Once daily        03-FEB-16   23-FEB-16   01-FEB-16
    6     15      6      7 105 mg   Once daily        18-OCT-14   02-DEC-16   14-OCT-14
    7     14     12     10 100 mg   Three times daily 08-JUL-14   20-AUG-14   04-JUL-14
    8     11     13      1 142 mg   Once daily        15-MAY-15   17-JUN-16   10-MAY-15
    9     15      4      4 55 mg    Twice daily       29-DEC-14   16-FEB-15   27-DEC-14
   10     11      7      2 56 mg    Twice daily       30-JUN-14   26-AUG-16   25-JUN-14
   11     13      1      4 39 mg    Three times daily 28-JAN-15   24-MAR-15   23-JAN-15
   12     12     13      8 85 mg    Once daily        14-JAN-15   13-MAR-16   13-JAN-15
   13     14      6      3 93 mg    Twice daily       24-APR-15   17-JUN-15   20-APR-15
   14     12      4      5 24 mg    Twice daily       04-AUG-15   17-SEP-15   03-AUG-15
   15     15      1      4 35 mg    Once daily        29-JUL-14   23-SEP-14   25-JUL-14
   16     13     12      5 136 mg   Once daily        07-JUN-14   05-AUG-14   04-JUN-14
   17     13      9      4 122 mg   Three times daily 23-APR-15   24-MAY-16   20-APR-15
   18     15      4      3 65 mg    Three times daily 24-JAN-15   09-MAR-16   22-JAN-15
   19     13     13      2 88 mg    Twice daily       12-FEB-14   07-MAR-16   11-FEB-14
   20     14      3      2 52 mg    Three times daily 22-AUG-14   17-OCT-16   18-AUG-14
   21     11     14      1 142 mg   Once daily        15-MAY-15   17-JUN-16   10-MAY-15

21 rows selected.

WINTER342 SQL> spool off
```

Medication:

```
WINTER342 SQL> desc arjt_medication
 Name                      Null?    Type
 ------------------------- -------- ----------------
 MED_ID                    NOT NULL NUMBER(5)
 MED_NAME                  NOT NULL VARCHAR2(30)
 MED_PURP                  NOT NULL VARCHAR2(30)

WINTER342 SQL> spool off
```

```
WINTER342 SQL> select * from arjt_medication;

    MED_ID MED_NAME                       MED_PURP
---------- ---------------------------- --------------------
         1 Triclosan                    Painkiller
         2 Acetaminophen                Blood thinner
         3 Cytarabine                   Insomnia
         4 Flurazepam Hydrochloride     Laxative
         5 Lisinopril                   Antibiotic
         6 Loratadine                   Antibiotic
         7 Ciprofloxacin                Blood thinner
         8 Hydrocodon                   Painkiller
         9 Escitalopram Oxalate         Blood thinner
        10 Spironolactone               Anti-allergin

10 rows selected.

WINTER342 SQL> spool off
```

Room:

```
WINTER342 SQL> desc arjt_room
 Name                      Null?    Type
 ------------------------- -------- ----------------
 ROOM_ID                   NOT NULL NUMBER(5)
 UNIT_ID                   NOT NULL NUMBER(5)
 ROOM_NO                   NOT NULL NUMBER(3)

WINTER342 SQL> spool off
```

```
WINTER342 SQL> select * from arjt_room;

ROOM_ID UNIT_ID ROOM_NO
------- ------- -------
      1       1     101
      2       1     102
      3       1     103
      4       1     104
      5       1     105
      6       1     106
      7       1     107
      8       2     108
      9       2     109
     10       2     110
     11       2     201
     12       2     202
     13       2     203
     14       2     204
     15       3     205
     16       3     206
     17       3     207
     18       3     208
     19       3     209
     20       3     210
     21       4     301
     22       4     302
     23       4     303
     24       4     304
     25       4     305
     26       5     306
     27       5     307
     28       5     308
     29       5     309
     30       5     310

30 rows selected.

WINTER342 SQL> spool off
```

Unit:

```
WINTER342 SQL> desc arjt_unit
 Name                      Null?    Type
 ------------------------- -------- ----------------
 UNIT_ID                   NOT NULL NUMBER(5)
 UNIT_NAME                 NOT NULL VARCHAR2(50)
 UNIT_ABBRV                NOT NULL VARCHAR2(10)

WINTER342 SQL> spool off
```

```
 WINTER342 SQL> select * from arjt_unit;

 UNIT_ID UNIT_NAME                    UNIT_ABBRV
 ------- ---------------------------- ----------
       1 Accute assessment            ACU
       2 Intensive care               ICU
       3 Nenoatal intensive care      NICU
       4 Physical therapy             PT
       5 Medical/Surgical             MedSurg

 WINTER342 SQL> spool off
```

Patient:

```
WINTER342 SQL> desc arjt_patient
 Name                      Null?    Type
 ------------------------- -------- ----------------
 PAT_ID                    NOT NULL NUMBER(5)
 SSN                       NOT NULL NUMBER(9)
 FIRST_NAME                NOT NULL VARCHAR2(30)
 MID_NAME                           VARCHAR2(30)
 LAST_NAME                 NOT NULL VARCHAR2(30)
 STREET                    NOT NULL VARCHAR2(50)
 ZIP                       NOT NULL NUMBER(5)
 CITY                      NOT NULL VARCHAR2(50)
 STATE                     NOT NULL VARCHAR2(50)
 PHONE                     NOT NULL NUMBER(10)
 DOB                       NOT NULL DATE
 GENDER                    NOT NULL VARCHAR2(1)
 INS_STAT                  NOT NULL VARCHAR2(50)
 LANGUAGE                  NOT NULL VARCHAR2(50)

WINTER342 SQL> spool off
```

```
WINTER342 SQL> select * from arjt_patient;

 PAT_ID        SSN FIRST_NAME MID_NAME   LAST_NAME  STREET                       ZIP CITY             STATE          PHONE      DOB       G INS_STAT    LANGUAGE
------- ---------- ---------- ---------- ---------- ------------------------ ------ ---------------- -------------- ---------- --------- - ----------- -------------
      1  465170825 Chloe      Terry      Price      1354 Valley Edge Alley    31231 Creil            Maryland       7637540760 28-SEP-15 F Medicare    English
      2  647780742 Max        Catherine  Caulfield  1035 Dapin Terrace        65576 Pirca            North Carolina 7285187157 14-DEC-15 F Medicare    French
      3  854015608 Kate       Juan       Marsh      9779 Stephen Pass         46774 Gaofeng          New York       6374340268 25-JUL-15 F Medicaid    English
      4  672636578 Lisa       The        Plant      789 Straubel Pass         95632 Memphis          Pennsylvania   6489663918 02-SEP-15 M None        Spanish
      5  169531477 William    George     Price      79658 Mockingbird Court   34807 Pandean          Colorado       2993860306 01-DEC-15 M None        English
      6  504794456 Warren     Nicole     Graham     1 Blaine Street           18044 Saint-Pie        Delaware       1412685790 24-JUN-15 M Medicaid    Japaneese
      7  264625346 Victoria   Ruby       Chase      76282 Northview Court     52169 Angelholm        Georgia        6776861920 12-FEB-15 F Medicare    Spanish
      8  960041017 Dana       Wayne      Ward       185 Bay Park              84421 Santa Gertrudes  California     4361094869 02-APR-15 F Medicaid    Japaneese
      9  441380586 Alyssa     Jeffrey    Anderson   22718 Fulton Parkway      20448 Salitral         Arizona        1899578244 12-JAN-16 F Medicare    Spanish
     10  499757261 Rachel     Ernest     Amber      9 Stang Court             96853 Ravenstone       California     8836628248 05-MAY-15 F None        Japaneese
     11  128458573 Frank      Jonathan   Bowers     5 Kipling Drive           20910 Doucard          Florida        4601547204 11-MAR-15 M Medicare    Spanish
     12  914594909 Stella     Rose       Hill       1804 Ridgeview Circle     51083 Pencilrock       Pennsylvania   7761872373 10-MAR-15 F None        Spanish
     13  371083748 Pompidou   The        Dog        5 Claremont Trail         97957 Papercisors      Tennessee      3085107873 17-JAN-16 M Medicare    Bark
     14  196547325 Luke       Thomas     Parker     35 Jackson Point          55423 Tulip            Connecticut    4979614660 24-FEB-15 F Medicaid    Japaneese
     15  878345785 Daniel     Sara       DaCosta    87 Farmco Junction        17043 Pertination      California     2914710251 05-FEB-16 M Medicaid    French
15 rows selected.
WINTER342 SQL> spool off
```

Placed:

```
WINTER342 SQL> desc arjt_placed
 Name                      Null?    Type
 ------------------------- -------- ----------------
 PAT_ID                    NOT NULL NUMBER(5)
 ROOM_ID                   NOT NULL NUMBER(5)
 START_DATE                NOT NULL DATE
 END_DATE                           DATE

WINTER342 SQL> spool off
```

```
WINTER342 SQL> select * from arjt_placed;

PAT_ID ROOM_ID START_DATE  END_DATE
------ ------- ----------- -----------
     3       1 21-DEC-15   17-JAN-16
     1       2 01-FEB-14   08-FEB-14
     2       2 02-FEB-14   10-FEB-14
     8      12 18-SEP-15   10-OCT-15
     2      17 11-JAN-16   07-FEB-16
     6      25 29-APR-15   18-MAY-15
     4      19 01-JAN-16   13-JAN-16
    10      26 15-FEB-15   07-MAR-15
     9      27 28-DEC-15   15-JAN-16
    11       2 16-OCT-15   01-NOV-15
    15      25 04-MAR-15   28-MAR-15
    13      25 26-JUL-15   09-AUG-15
    12      21 24-MAR-15   06-APR-15
     5      21 23-SEP-15   22-OCT-15
     7      15 21-AUG-15   02-SEP-15
     6       2 18-MAY-15   06-JUN-15
     4      23 13-JAN-16   11-FEB-16
    10      14 07-MAR-15   02-APR-15
     9      17 15-JAN-16   06-FEB-16
    11       7 01-NOV-15   22-NOV-15
    15       4 28-MAR-15   24-APR-15
    13      29 09-AUG-15   04-SEP-15
    12       3 06-APR-15   25-APR-15
     5      28 22-OCT-15   17-NOV-15
     7      17 02-SEP-15   01-OCT-15
     6      13 06-JUN-15   26-JUN-15
     4      23 11-FEB-16   03-MAR-16
    10       6 02-APR-15   12-APR-15
     9      18 06-FEB-16   06-MAR-16
    11      11 22-NOV-15   05-DEC-15
    15      12 24-APR-15   21-MAY-15
    13      15 04-SEP-15   26-SEP-15
    12      21 25-APR-15   23-MAY-15
     5      12 17-NOV-15   02-DEC-15
     7      28 01-OCT-15   25-OCT-15
     6       6 26-JUN-15   06-JUL-15
     4      29 03-MAR-16   01-APR-16
    10       8 12-APR-15   12-MAY-15
     9      12 06-MAR-16   19-MAR-16
    11      14 05-DEC-15   02-JAN-16
    15      12 21-MAY-15   03-JUN-15
```

```
    13        8 26-SEP-15     20-OCT-15
    12       28 23-MAY-15     18-JUN-15
     5       20 02-DEC-15     29-DEC-15
     7       18 25-OCT-15     17-NOV-15
     6       22 06-JUL-15     05-AUG-15
     4       14 01-APR-16     16-APR-16
    10       22 12-MAY-15     08-JUN-15
     9       25 19-MAR-16     18-APR-16
    11       17 02-JAN-16     18-JAN-16
    15        1 03-JUN-15     29-JUN-15
    13       20 20-OCT-15     13-NOV-15
    12       20 18-JUN-15     01-JUL-15
     5        9 29-DEC-15     25-JAN-16
     7        3 17-NOV-15     10-DEC-15
     6       11 05-AUG-15     27-AUG-15
     4       23 16-APR-16     27-APR-16
    10       29 08-JUN-15     21-JUN-15
     9       13 18-APR-16     02-MAY-16
    11       30 18-JAN-16     17-FEB-16
    15        7 29-JUN-15     19-JUL-15
    13       12 13-NOV-15     13-DEC-15
    12       23 01-JUL-15     20-JUL-15
     5        2 25-JAN-16     10-FEB-16
     7       15 10-DEC-15     24-DEC-15
     6       13 27-AUG-15     24-SEP-15
     4       28 27-APR-16     16-MAY-16
    10       20 21-JUN-15     08-JUL-15
     9       26 02-MAY-16     17-MAY-16
    11       25 17-FEB-16     04-MAR-16
    15       22 19-JUL-15     16-AUG-15
    13       24 13-DEC-15     11-JAN-16
    12       11 20-JUL-15     03-AUG-15
     5       18 10-FEB-16     27-FEB-16
     7       27 24-DEC-15     09-JAN-16
     6       29 24-SEP-15     17-OCT-15
     4       24 16-MAY-16     27-MAY-16
    10        5 08-JUL-15     06-AUG-15
     9       16 17-MAY-16     16-JUN-16
    11       29 04-MAR-16     21-MAR-16

80 rows selected.

WINTER342 SQL> spool off
```

Patient Admitted:

```
WINTER342 SQL> desc arjt_patient_admitted
 Name                      Null?    Type
 ------------------------- -------- ----------------
 PAT_ID                    NOT NULL NUMBER(5)
 ADM_DATE                  NOT NULL DATE
 DIS_DATE                           DATE

WINTER342 SQL> spool off
```

```
WINTER342 SQL> select * from arjt_patient_admitted;

PAT_ID ADM_DATE  DIS_DATE
------ --------- ---------
     3 21-DEC-15 17-JAN-16
     1 25-MAY-15 07-JUN-15
    14 09-JAN-16 04-FEB-16
     8 18-SEP-15 10-OCT-15
     2 11-JAN-16 07-FEB-16
     6 29-APR-15 18-MAY-15
     4 01-JAN-16 13-JAN-16
    10 15-FEB-15 07-MAR-15
     9 28-DEC-15 15-JAN-16
    11 16-OCT-15 01-NOV-15
    15 04-MAR-15 28-MAR-15
    13 26-JUL-15 09-AUG-15
    12 24-MAR-15 06-APR-15
     5 23-SEP-15 22-OCT-15
     7 21-AUG-15 02-SEP-15
     6 18-MAY-15 06-JUN-15
     4 13-JAN-16 11-FEB-16
    10 07-MAR-15 02-APR-15
     9 15-JAN-16 06-FEB-16
    11 01-NOV-15 22-NOV-15
    15 28-MAR-15 24-APR-15
    13 09-AUG-15 04-SEP-15
    12 06-APR-15 25-APR-15
     5 22-OCT-15 17-NOV-15
     7 02-SEP-15 01-OCT-15
     6 06-JUN-15 26-JUN-15
     4 11-FEB-16 03-MAR-16
    10 02-APR-15 12-APR-15
     9 06-FEB-16 06-MAR-16
    11 22-NOV-15 05-DEC-15
    15 24-APR-15 21-MAY-15
    13 04-SEP-15 26-SEP-15
    12 25-APR-15 23-MAY-15
     5 17-NOV-15 02-DEC-15
     7 01-OCT-15 25-OCT-15
     6 26-JUN-15 06-JUL-15
     4 03-MAR-16 01-APR-16
    10 12-APR-15 12-MAY-15
     9 06-MAR-16 19-MAR-16
    11 05-DEC-15 02-JAN-16
    15 21-MAY-15 03-JUN-15
```

```
     12 23-MAY-15 18-JUN-15
      5 02-DEC-15 29-DEC-15
      7 25-OCT-15 17-NOV-15
      6 06-JUL-15 05-AUG-15
      4 01-APR-16 16-APR-16
     10 12-MAY-15 08-JUN-15
      9 19-MAR-16 18-APR-16
     11 02-JAN-16 18-JAN-16
     15 03-JUN-15 29-JUN-15
     13 20-OCT-15 13-NOV-15
     12 18-JUN-15 01-JUL-15
      5 29-DEC-15 25-JAN-16
      7 17-NOV-15 10-DEC-15
      6 05-AUG-15 27-AUG-15
      4 16-APR-16 27-APR-16
     10 08-JUN-15 21-JUN-15
      9 18-APR-16 02-MAY-16
     11 18-JAN-16 17-FEB-16
     15 29-JUN-15 19-JUL-15
     13 13-NOV-15 13-DEC-15
     12 01-JUL-15 20-JUL-15
      5 25-JAN-16 10-FEB-16
      7 10-DEC-15 24-DEC-15
      6 27-AUG-15 24-SEP-15
      4 27-APR-16 16-MAY-16
     10 21-JUN-15 08-JUL-15
      9 02-MAY-16 17-MAY-16
     11 17-FEB-16 04-MAR-16
     15 19-JUL-15
     13 13-DEC-15
     12 20-JUL-15
      5 10-FEB-16
      7 24-DEC-15
      6 24-SEP-15
      4 16-MAY-16
     10 08-JUL-15
      9 17-MAY-16
      1 04-MAR-16
     11 04-MAR-16
      8 04-MAR-16

82 rows selected.

WINTER342 SQL> spool off
```

Assessment:

```
WINTER342 SQL> desc arjt_assessment
 Name                      Null?    Type
 ------------------------- -------- ----------------
 ASMT_ID                   NOT NULL NUMBER(5)
 EMP_ID                    NOT NULL NUMBER(5)
 PAT_ID                    NOT NULL NUMBER(5)
 BP_SYS                    NOT NULL NUMBER(3)
 BP_DIA                    NOT NULL NUMBER(3)
 RESP_RATE                 NOT NULL NUMBER(3)
 HRT_RATE                  NOT NULL NUMBER(3)
 DAT                       NOT NULL TIMESTAMP(6)

WINTER342 SQL> spool off
```

```
WINTER342 SQL> select * from arjt_assessment;

ASMT_ID EMP_ID PAT_ID BP_SYS BP_DIA RESP_RATE HRT_RATE DAT
------- ------ ------ ------ ------ --------- -------- ---------------------------
      1      5      1    177     99        60      151 17-OCT-15 01.42.27.000000 PM
      2      9     15    119     56        43       91 19-DEC-15 04.38.27.000000 PM
      3      1      9    199     75        10      111 11-MAR-15 02.08.09.000000 PM
      4      9      6    125     98        10      149 16-JUL-15 12.00.47.000000 AM
      5      6      4    120     81        39       46 19-JUN-15 08.43.26.000000 AM
      6      6     10    137     38         9      119 22-NOV-15 03.33.20.000000 PM
      7     10     11    166     83        31       36 25-MAY-15 07.55.31.000000 AM
      8      6      1    107     96        42       63 10-SEP-15 06.16.20.000000 AM
      9      7      4    105     80        32      148 04-AUG-15 07.45.29.000000 AM
     10      4      6    180     55        15      121 28-AUG-15 02.41.12.000000 PM
     11      7      2    140     77        37      179 09-NOV-15 09.49.41.000000 PM
     12      5     13    194     43        13       23 12-DEC-15 03.43.03.000000 PM
     13      2      5    110     84        23      107 25-JAN-16 09.20.48.000000 PM
     14      6      1    109     60        54      106 29-APR-15 12.46.21.000000 PM
     15     10      4    190     35        28      104 22-JUL-15 06.42.29.000000 AM
     16      7      4    166     94        38       95 14-JUN-15 03.36.41.000000 AM
     17      4      4    165     38        54       58 16-DEC-15 01.52.21.000000 PM
     18      6      5    178     42        49       44 18-FEB-15 06.52.48.000000 AM
     19      1     11    116     96        28       83 30-NOV-15 12.56.17.000000 AM
     20      1      8    126     71         8      145 10-DEC-15 10.00.53.000000 AM
     21      7      7    150     92        59      150 08-JUL-15 11.44.48.000000 AM
     22      1     10    121     62        42      165 07-FEB-16 12.52.30.000000 PM
     23      1      7    158     54        21      103 03-JAN-16 05.52.54.000000 PM
     24     10      6    168     51        13      176 12-OCT-15 09.55.34.000000 PM
     25      5      1    172     55        10      158 22-MAY-15 09.42.59.000000 PM
     26      5     10    138     35         9      118 21-FEB-16 01.35.14.000000 AM
     27     10      4    177     32        39      129 19-NOV-15 12.10.24.000000 PM
     28      9     13    152     57        33      118 03-APR-15 10.19.07.000000 AM
     29      5      8    197     85        48      112 18-NOV-15 04.05.23.000000 PM
     30      2      1    130     30        45      104 20-OCT-15 02.54.37.000000 AM
     31      2     12    184     85        59      111 07-DEC-15 05.39.24.000000 AM
     32      4      3    197     44        22       96 30-JUL-15 09.40.44.000000 AM
     33      8     15    135     74        28      103 26-DEC-15 08.26.45.000000 AM
     34     10      2    160     86        29       41 09-AUG-15 11.03.31.000000 PM
     35      6      5    123     66        53       43 18-JUN-15 03.18.38.000000 AM
     36      3     13    189     61        11      124 05-JUL-15 01.54.22.000000 AM
     37      8      7    105     56        47       75 01-MAY-15 06.41.46.000000 AM
     38      4     10    122     44        27       35 03-APR-15 12.54.36.000000 AM
     39      7      1    159     62         6      104 31-DEC-15 08.06.33.000000 PM
     40      9     12    200     44        37      115 01-JAN-16 07.49.28.000000 AM
     41      6     15    113     40        14      145 18-SEP-15 05.38.15.000000 AM
     42      7     10    171     48        25       42 12-JAN-16 09.56.54.000000 AM
     43     10      7    172     82        11      104 07-NOV-15 09.52.03.000000 AM
     44      6      9    153     93         5      104 22-DEC-15 02.27.06.000000 AM
     45      6      3    197     78        51      167 10-DEC-15 06.37.13.000000 PM
     46      7     15    105     74        60       51 10-DEC-15 03.13.35.000000 AM
     47      3      9    105     72        24       62 14-FEB-15 09.14.38.000000 PM
     48      1      2    199     57        28       94 05-OCT-15 11.02.33.000000 PM
     49      8     10    137     44        17      103 12-JUL-15 04.23.17.000000 AM
     50      3      1    102     67        15      125 24-DEC-15 03.35.14.000000 PM
```

```
        51          7         14        116         34         45         90 22-OCT-15 05.28.06.000000 PM
        52         10          2        173         48         55         76 02-JUL-15 07.33.11.000000 AM
        53          6          5        162         30         13         90 05-DEC-15 05.33.49.000000 PM
        54         10          8        169         56         50         85 26-APR-15 06.44.52.000000 AM
        55          2          4        185         84         15        172 27-SEP-15 07.04.01.000000 PM
        56         10          2        168         70         38        103 22-FEB-15 10.52.48.000000 PM
        57          6          5        193         88          6        168 15-MAY-15 05.16.19.000000 PM
        58          1          6        180         90          9        147 18-JUN-15 10.29.33.000000 PM
        59          6          3        174         82         51        168 22-FEB-15 03.56.08.000000 PM
        60          5          4        176         84         43        174 29-OCT-15 01.51.45.000000 PM
        61          4          4        176         84         43        174 30-OCT-15 01.51.45.000000 PM
        62          3          4        176         84         43        174 31-OCT-15 01.51.45.000000 PM
        63          2          4        176         84         43        174 01-OCT-15 01.51.45.000000 PM
        64          1          4        176         84         43        174 02-OCT-15 01.51.45.000000 PM
        65          7          4        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        66          1          1        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        67          2          1        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        68          3          1        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        69          4          1        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        70          5          1        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        71          7          1        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        72          1          8        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        73          2          8        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        74          3          8        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        75          4          8        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        76          5          8        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        77          7          8        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        78          6          8        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        79          1          5        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        80          2          5        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        81          3          5        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        82          4          5        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        83          5          5        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        84          6          5        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        85          7          5        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        86          1         10        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        87          2         10        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        88          3         10        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        89          4         10        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        90          5         10        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        91          7         10        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        92          7         10        176         84         43        174 03-OCT-15 01.51.45.000000 PM
        93          5          2        138         35          9        118 21-FEB-16 01.35.14.000000 AM
        94          5          6        138         35          9        118 21-FEB-16 01.35.14.000000 AM
        95          5          8        138         35          9        118 21-FEB-16 01.35.14.000000 AM

95 rows selected.

WINTER342 SQL> spool off
```

Special Need:

```
WINTER342 SQL> desc arjt_special_need
 Name                      Null?    Type
 ------------------------- -------- ----------------
 SN_ID                     NOT NULL NUMBER(5)
 EMP_ID                    NOT NULL NUMBER(5)
 PAT_ID                    NOT NULL NUMBER(5)
 ACT_ID                    NOT NULL NUMBER(5)
 FREQ                      NOT NULL VARCHAR2(100)
 START_DATE                NOT NULL DATE
 END_DATE                  NOT NULL DATE
 DAT                       NOT NULL DATE

WINTER342 SQL> spool off
```

```
WINTER342 SQL> select * from arjt_special_need;

SN_ID EMP_ID PAT_ID     ACT_ID FREQ                START_DATE   END_DATE     DAT
----- ------ ------ ---------- ------------------- ------------ ------------ ------------
    1     11      9          9 Once daily          07-FEB-16    04-MAR-16    07-FEB-16
    2     12     11          4 Three times daily   24-NOV-15    03-DEC-15    24-NOV-15
    3     11     15          8 Once daily          25-APR-15    19-MAY-15    25-APR-15
    4     11     13          7 Twice daily         05-SEP-15    24-SEP-15    05-SEP-15
    5     14     12          3 Twice daily         26-APR-15    22-MAY-15    26-APR-15
    6     11      5          5 Three times daily   19-NOV-15    01-DEC-15    19-NOV-15
    7     12      7         10 Once daily          03-OCT-15    24-OCT-15    03-OCT-15
    8     13      6          2 Twice daily         27-JUN-15    05-JUL-15    27-JUN-15
    9     14      4          8 Twice daily         05-MAR-16    31-MAR-16    05-MAR-16
   10     14     10         10 Once daily          14-APR-15    11-MAY-15    14-APR-15
   11     12      9          6 Once daily          08-MAR-16    17-MAR-16    08-MAR-16
   12     12     11          1 Twice daily         06-DEC-15    31-DEC-15    06-DEC-15
   13     11     15          7 Twice daily         23-MAY-15    01-JUN-15    23-MAY-15
   14     15     13          7 Three times daily   27-SEP-15    19-OCT-15    27-SEP-15
   15     12     12          2 Once daily          24-MAY-15    16-JUN-15    24-MAY-15
   16     14      5          8 Twice daily         03-DEC-15    28-DEC-15    03-DEC-15
   17     14      7          2 Twice daily         26-OCT-15    16-NOV-15    26-OCT-15
   18     12      6          6 Once daily          07-JUL-15    04-AUG-15    07-JUL-15
   19     15      4          8 Twice daily         03-APR-16    15-APR-16    03-APR-16
   20     11     10          8 Once daily          14-MAY-15    07-JUN-15    14-MAY-15
   21     12      2          8 Once daily          14-MAY-15    07-JUN-15    14-MAY-15
   22     12      2          9 Once daily          14-MAY-15    07-JUN-15    14-MAY-15
   23     15      3          8 Once daily          14-MAY-15    07-JUN-15    14-MAY-15
   24     15      3          9 Once daily          14-MAY-15    07-JUN-15    14-MAY-15
   25     11      4          8 Once daily          14-MAY-15    07-JUN-15    14-MAY-15
   26     11      4          9 Once daily          14-MAY-15    07-JUN-15    14-MAY-15

26 rows selected.

WINTER342 SQL> spool off
```

Activity:

```
WINTER342 SQL> desc arjt_activity
 Name                      Null?    Type
 ------------------------- -------- ----------------
 ACT_ID                    NOT NULL NUMBER(5)
 ACT_NAME                  NOT NULL VARCHAR2(30)
 DESCR                              VARCHAR2(70)

WINTER342 SQL> spool off
```

```
WINTER342 SQL> select * from arjt_activity;

ACT_ID ACT_NAME                   DESCR
------ -------------------------
---------------------------------------
     1 bed turn                   Rotate patient in bed
     2 wound care                 Clean and dress wounds
     3 catheter care              Change out catheter
     4 showering                  Assist patient with shower
     5 burn care                  Clean and care for burn wounds
     6 physical therapy           Perform physical therapy for patient
     7 functional mobility        Help patient walk and move about
     8 assisted feeding           Help patient consume food
     9 assisted toilet hygiene    Help patient use the bathroom
    10 bathing                    Assist patient with bath

10 rows selected.

WINTER342 SQL> spool off
```

Administers:

```
WINTER342 SQL> desc arjt_administers
 Name                      Null?    Type
 ------------------------- -------- ----------------
 EMP_ID                    NOT NULL NUMBER(5)
 RX_ID                     NOT NULL NUMBER(5)
 DAT                       NOT NULL TIMESTAMP(6)

WINTER342 SQL> spool off
```

```
WINTER342 SQL> select * from arjt_administers;

EMP_ID RX_ID DAT
------ ----- ------------------------------
     1     1 22-NOV-15 02.26.44.000000 PM
     1     2 13-MAY-15 09.31.27.000000 PM
     1     2 21-JUN-15 03.27.58.000000 PM
     1     3 01-MAY-15 03.42.03.000000 PM
     1     3 18-AUG-15 07.02.25.000000 AM
     1     5 07-MAR-15 11.59.21.000000 PM
     1     9 29-JAN-16 07.20.14.000000 PM
     1    13 29-JUN-15 02.59.44.000000 PM
     1    15 10-JUN-15 09.46.01.000000 PM
     1    16 25-MAY-15 07.41.34.000000 AM
     1    18 29-JUN-15 05.34.31.000000 PM
     1    19 18-SEP-15 07.46.49.000000 PM
     2     4 08-JUL-15 03.08.54.000000 PM
     2     5 15-MAY-15 07.42.00.000000 AM
     2     6 08-DEC-15 05.27.51.000000 PM
     2     9 06-NOV-15 04.15.34.000000 PM
     2    12 04-JAN-16 03.52.28.000000 AM
     2    14 08-MAR-15 07.40.16.000000 AM
     2    14 26-JAN-16 07.02.34.000000 PM
     2    16 02-MAY-15 08.25.41.000000 PM
     3     3 24-FEB-15 01.49.18.000000 PM
     3     3 04-MAY-15 04.30.24.000000 AM
     3     4 16-DEC-15 12.45.41.000000 AM
     3     6 08-AUG-15 11.33.04.000000 AM
     3     7 20-JUN-15 02.47.33.000000 AM
     3    17 19-NOV-15 02.21.26.000000 PM
     4     8 03-JAN-16 07.17.40.000000 AM
     4    16 28-AUG-15 12.15.13.000000 AM
     4    18 09-SEP-15 04.12.37.000000 AM
     5     3 18-APR-15 01.10.35.000000 AM
     5     3 26-JUN-15 07.48.40.000000 PM
     5     4 09-FEB-16 05.54.53.000000 AM
     5     7 24-MAY-15 11.30.36.000000 PM
     5    11 30-JUN-15 09.24.33.000000 PM
     5    12 11-FEB-16 02.03.24.000000 PM
     5    13 16-JUN-15 04.30.36.000000 AM
     5    13 24-NOV-15 01.53.40.000000 PM
     5    14 21-JUL-15 10.47.28.000000 PM
     5    15 09-APR-15 12.00.33.000000 PM
     5    15 10-OCT-15 09.57.02.000000 PM
     5    19 30-MAR-15 08.40.58.000000 PM
```

```
     6      1 16-AUG-15 01.50.07.000000 AM
     6      2 10-MAY-15 02.35.03.000000 AM
     6      3 12-MAR-15 04.32.58.000000 AM
     6      4 11-NOV-15 09.17.45.000000 PM
     6      5 25-FEB-15 10.12.10.000000 AM
     6      6 25-MAY-15 12.10.58.000000 AM
     6      8 06-DEC-15 02.19.05.000000 AM
     6      9 08-JUL-15 08.33.31.000000 AM
     6     11 11-DEC-15 02.01.04.000000 PM
     7      4 29-SEP-15 08.07.36.000000 AM
     7     11 20-JUN-15 03.53.07.000000 PM
     7     12 01-MAR-15 10.12.44.000000 PM
     7     13 08-SEP-15 07.51.07.000000 PM
     7     15 28-FEB-15 05.34.47.000000 PM
     7     19 17-NOV-15 05.39.47.000000 AM
     8      5 21-OCT-15 08.05.13.000000 PM
     8      6 16-FEB-15 07.58.25.000000 AM
     8      6 22-MAR-15 10.26.11.000000 PM
     8      8 25-NOV-15 03.50.01.000000 PM
     8      9 23-MAY-15 11.11.13.000000 PM
     8     15 19-MAR-15 05.55.04.000000 AM
     8     17 21-JUN-15 12.10.34.000000 AM
     8     20 05-MAY-15 11.02.49.000000 AM
     9      3 28-JAN-16 06.21.31.000000 AM
     9      5 10-JUL-15 01.02.53.000000 PM
     9      7 19-FEB-15 06.17.36.000000 PM
     9      7 12-APR-15 08.14.42.000000 PM
     9      8 25-AUG-15 05.22.29.000000 AM
     9     13 14-MAR-15 11.38.43.000000 AM
    10      1 22-FEB-15 04.19.47.000000 PM
    10      3 26-JAN-16 03.14.10.000000 AM
    10      6 16-FEB-15 04.59.07.000000 AM
    10      7 25-NOV-15 09.14.34.000000 PM
    10      8 10-AUG-15 03.45.23.000000 PM
    10      9 26-MAR-15 12.50.52.000000 PM
    10     10 30-MAR-15 12.18.03.000000 PM
    10     11 09-DEC-15 03.30.58.000000 AM
    10     13 29-DEC-15 02.49.38.000000 PM
    10     14 25-JUL-15 06.55.11.000000 PM

80 rows selected.

WINTER342 SQL> spool off
```

Assigned To:

```
WINTER342 SQL> desc arjt_assigned_to
 Name                      Null?    Type
 ------------------------- -------- ----------------
 EMP_ID                    NOT NULL NUMBER(5)
 PAT_ID                    NOT NULL NUMBER(5)
 START_DATE                NOT NULL TIMESTAMP(6)
 END_DATE                           TIMESTAMP(6)

WINTER342 SQL> spool off
```

```
WINTER342 SQL> select * from arjt_assigned_to;

EMP_ID PAT_ID START_DATE                         END_DATE
------ ------ ------------------------------ ------------------------------
     9     10 10-FEB-16 06.56.00.000000 PM   11-FEB-16 11.56.00.000000 AM
     5      1 04-OCT-15 12.37.00.000000 PM   05-OCT-15 04.37.00.000000 AM
     2      2 07-DEC-15 12.07.00.000000 AM   07-DEC-15 09.07.00.000000 PM
     5      2 30-JUN-15 09.47.00.000000 PM   01-JUL-15 08.47.00.000000 PM
    10      2 13-JAN-16 01.39.00.000000 AM   13-JAN-16 02.39.00.000000 PM
    10      5 16-FEB-15 09.47.00.000000 PM   17-FEB-15 07.47.00.000000 PM
     4      7 02-JUL-15 05.09.00.000000 PM   02-JUL-15 05.09.00.000000 PM
     1      2 08-OCT-15 02.04.00.000000 PM   09-OCT-15 09.04.00.000000 AM
     1      2 03-JAN-16 08.00.00.000000 PM   04-JAN-16 07.00.00.000000 PM
     9      2 26-AUG-15 07.50.00.000000 PM   27-AUG-15 01.50.00.000000 PM
     2      3 06-JUL-15 05.07.00.000000 PM   07-JUL-15 01.07.00.000000 PM
     2     12 06-JUN-15 08.57.00.000000 AM   07-JUN-15 07.57.00.000000 AM
     9     12 15-MAR-15 07.48.00.000000 AM   16-MAR-15 05.48.00.000000 AM
     1      7 09-OCT-15 07.30.00.000000 AM   10-OCT-15 01.30.00.000000 AM
     1     14 08-AUG-15 10.56.00.000000 AM   09-AUG-15 03.56.00.000000 AM
     2      4 02-MAR-15 03.36.00.000000 AM   02-MAR-15 05.36.00.000000 PM
     8      3 27-SEP-15 05.12.00.000000 PM   28-SEP-15 03.12.00.000000 PM
     7     13 10-NOV-15 10.22.00.000000 PM   11-NOV-15 02.22.00.000000 PM
     5     12 25-APR-15 06.48.00.000000 AM   25-APR-15 09.48.00.000000 PM
    10      2 13-SEP-15 07.37.00.000000 AM   14-SEP-15 03.37.00.000000 AM
     2     12 25-SEP-15 01.22.00.000000 AM   25-SEP-15 05.22.00.000000 PM
     2     11 18-JAN-16 05.24.00.000000 PM   19-JAN-16 04.24.00.000000 PM
     8      7 16-OCT-15 06.51.00.000000 AM   16-OCT-15 10.51.00.000000 PM
     4      7 13-NOV-15 05.07.00.000000 AM   14-NOV-15 04.07.00.000000 AM
     4      1 27-JUL-15 08.54.00.000000 PM   28-JUL-15 02.54.00.000000 PM
     3     13 16-FEB-15 02.09.00.000000 PM   17-FEB-15 01.09.00.000000 PM
     4     10 29-DEC-15 08.12.00.000000 PM   30-DEC-15 05.12.00.000000 PM
     9     13 28-FEB-15 10.28.00.000000 AM   28-FEB-15 11.28.00.000000 PM
     9      7 24-NOV-15 05.41.00.000000 AM   24-NOV-15 09.41.00.000000 PM
     7     13 26-AUG-15 03.51.00.000000 AM   26-AUG-15 08.51.00.000000 PM
     4     13 18-DEC-15 06.09.00.000000 PM   19-DEC-15 12.09.00.000000 PM
     2     11 20-JUN-15 09.46.00.000000 PM   21-JUN-15 02.46.00.000000 PM
     8      6 25-OCT-15 06.11.00.000000 PM   26-OCT-15 03.11.00.000000 PM
     3      2 23-MAY-15 05.29.00.000000 PM   24-MAY-15 08.29.00.000000 AM
     3     15 04-NOV-15 04.01.00.000000 AM   04-NOV-15 11.01.00.000000 PM
     9      3 28-OCT-15 01.27.00.000000 AM   28-OCT-15 08.27.00.000000 PM
     6      9 01-MAY-15 06.37.00.000000 AM   02-MAY-15 05.37.00.000000 AM
     6      1 16-JUL-15 03.31.00.000000 AM   16-JUL-15 11.31.00.000000 PM
     7     10 17-MAR-15 11.25.00.000000 PM   18-MAR-15 03.25.00.000000 PM
     8      8 30-AUG-15 03.43.00.000000 AM   30-AUG-15 04.43.00.000000 PM
     5     13 18-JUN-15 07.40.00.000000 PM   19-JUN-15 02.40.00.000000 PM
     8      2 08-NOV-15 11.07.00.000000 PM   09-NOV-15 12.07.00.000000 PM
    10     15 16-DEC-15 05.00.00.000000 AM   17-DEC-15 01.00.00.000000 AM
```

```
 10      1 05-APR-15 11.28.00.000000 PM   06-APR-15 12.28.00.000000 PM
  9     11 22-OCT-15 09.17.00.000000 PM   23-OCT-15 11.17.00.000000 AM
  5      2 15-SEP-15 10.25.00.000000 PM   16-SEP-15 07.25.00.000000 PM
  1      4 30-APR-15 03.12.00.000000 PM   01-MAY-15 11.12.00.000000 AM
  8     13 21-DEC-15 05.23.00.000000 PM   22-DEC-15 09.23.00.000000 AM
  1      1 16-SEP-15 07.36.00.000000 AM   16-SEP-15 07.36.00.000000 PM
  7      1 07-APR-15 05.13.00.000000 AM   07-APR-15 07.13.00.000000 PM
  7      5 07-APR-15 07.20.00.000000 PM   08-APR-15 12.20.00.000000 PM
  2     10 16-AUG-15 10.32.00.000000 PM   17-AUG-15 06.32.00.000000 PM
  3     10 21-AUG-15 11.01.00.000000 AM   22-AUG-15 06.01.00.000000 AM
  7      9 11-MAR-15 10.58.00.000000 AM   12-MAR-15 01.58.00.000000 AM
  1      4 06-DEC-15 10.33.00.000000 PM   07-DEC-15 03.33.00.000000 PM
  9      1 06-SEP-15 04.34.00.000000 AM   06-SEP-15 04.34.00.000000 PM
  8      4 22-SEP-15 02.20.00.000000 AM   22-SEP-15 10.20.00.000000 PM
  3      9 05-NOV-15 06.10.00.000000 PM   06-NOV-15 04.10.00.000000 PM
  4      9 23-SEP-15 12.41.00.000000 AM   23-SEP-15 12.41.00.000000 AM
  6      9 09-MAR-15 02.10.00.000000 PM   10-MAR-15 07.10.00.000000 AM
  6      8 13-AUG-15 09.19.00.000000 AM   13-AUG-15 10.19.00.000000 PM
  4      8 15-FEB-15 03.15.00.000000 AM   16-FEB-15 01.15.00.000000 AM
  4      9 20-MAY-15 09.03.00.000000 AM   21-MAY-15 05.03.00.000000 AM
  1      4 06-DEC-15 11.01.00.000000 PM   07-DEC-15 09.01.00.000000 PM
  3     15 02-FEB-16 07.08.00.000000 PM   03-FEB-16 12.08.00.000000 PM
  9     10 25-NOV-15 08.29.00.000000 PM   26-NOV-15 01.29.00.000000 PM
  1      8 21-OCT-15 02.47.00.000000 AM   22-OCT-15 12.47.00.000000 AM
  1      2 20-OCT-15 05.51.00.000000 PM   21-OCT-15 09.51.00.000000 AM
  8      6 02-APR-15 11.29.00.000000 AM   03-APR-15 09.29.00.000000 AM
  9     11 07-AUG-15 11.24.00.000000 AM   08-AUG-15 07.24.00.000000 AM
  8      8 23-JUN-15 03.31.00.000000 AM   23-JUN-15 08.31.00.000000 PM
  6     14 11-OCT-15 05.24.00.000000 AM   12-OCT-15 02.24.00.000000 AM
  5      5 07-APR-15 03.37.00.000000 AM   07-APR-15 10.37.00.000000 PM
  9     14 10-JUL-15 06.12.00.000000 PM   11-JUL-15 06.12.00.000000 AM
  1     15 03-SEP-15 02.40.00.000000 PM   04-SEP-15 06.40.00.000000 AM
  5      4 17-JUN-15 09.58.00.000000 PM   18-JUN-15 11.58.00.000000 AM
  5      3 17-JUL-15 04.57.00.000000 PM   18-JUL-15 04.57.00.000000 AM
  4      2 19-MAR-15 02.22.00.000000 AM   19-MAR-15 11.22.00.000000 PM
  1      1 24-JUN-15 04.54.00.000000 AM   24-JUN-15 05.54.00.000000 PM
  5      8 15-FEB-15 01.34.00.000000 AM   15-FEB-15 04.34.00.000000 PM
  5      1 15-FEB-15 01.34.00.000000 AM   15-FEB-15 04.34.00.000000 PM
  6      2 15-FEB-15 01.34.00.000000 AM   15-FEB-15 04.34.00.000000 PM
  6      3 15-FEB-15 01.34.00.000000 AM   15-FEB-15 04.34.00.000000 PM
  7      2 15-FEB-15 01.34.00.000000 AM   15-FEB-15 04.34.00.000000 PM
  7      3 15-FEB-15 01.34.00.000000 AM   15-FEB-15 04.34.00.000000 PM

85 rows selected.

WINTER342 SQL> spool off
```

Performs:

```
WINTER342 SQL> desc arjt_performs
 Name                      Null?    Type
 ------------------------- -------- ----------------
 EMP_ID                    NOT NULL NUMBER(5)
 SN_ID                     NOT NULL NUMBER(5)
 DAT                       NOT NULL TIMESTAMP(6)

WINTER342 SQL> spool off
```

```
WINTER342 SQL> select * from arjt_performs;

EMP_ID SN_ID DAT
------ ----- ---------------------------
     1     1 18-SEP-15 09.54.47.000000 AM
     1     2 08-AUG-15 01.35.43.000000 PM
     1     5 23-MAR-15 10.39.11.000000 PM
     1     6 17-MAY-15 04.23.29.000000 PM
     1     6 10-AUG-15 06.54.18.000000 AM
     1    10 17-JUN-15 03.41.04.000000 AM
     1    11 08-JUN-15 01.36.22.000000 PM
     1    11 10-AUG-15 05.47.40.000000 PM
     1    12 28-FEB-15 07.17.53.000000 AM
     1    13 07-MAR-15 04.41.45.000000 AM
     1    15 24-NOV-15 11.41.14.000000 AM
     1    16 08-NOV-15 12.55.15.000000 PM
     1    17 05-OCT-15 11.57.58.000000 PM
     1    19 18-AUG-15 10.41.50.000000 PM
     2     1 18-SEP-15 09.54.47.000000 AM
     2     2 08-AUG-15 01.35.43.000000 PM
     2     4 26-JUN-15 11.21.25.000000 AM
     2     4 04-SEP-15 07.42.26.000000 PM
     2     5 23-MAR-15 10.39.11.000000 PM
     2     6 29-MAY-15 04.51.39.000000 PM
     2     6 10-AUG-15 06.54.18.000000 AM
     2    10 17-JUN-15 03.41.04.000000 AM
     2    10 08-FEB-16 04.18.33.000000 AM
     2    11 08-JUN-15 01.36.22.000000 PM
     2    11 30-NOV-15 04.49.15.000000 PM
     2    12 28-FEB-15 07.17.53.000000 AM
     2    13 07-MAR-15 04.41.45.000000 AM
     2    13 13-NOV-15 07.05.13.000000 PM
     2    16 24-AUG-15 03.32.45.000000 PM
     2    16 08-NOV-15 12.55.15.000000 PM
     2    17 16-MAY-15 08.05.56.000000 PM
     2    17 05-OCT-15 11.57.58.000000 PM
     2    20 24-NOV-15 05.31.04.000000 PM
     3     4 08-AUG-15 09.02.18.000000 PM
     3     4 25-NOV-15 09.07.50.000000 PM
     3     6 13-AUG-15 06.47.54.000000 AM
     3    13 20-JUL-15 08.47.42.000000 AM
     3    15 06-MAR-15 06.07.17.000000 PM
     3    18 13-JUN-15 01.03.26.000000 PM
     4     2 26-FEB-15 07.42.02.000000 AM
     4     2 13-MAY-15 06.17.22.000000 PM
     4     4 26-AUG-15 10.09.31.000000 PM
     4     8 23-SEP-15 11.40.02.000000 PM
     4    12 15-AUG-15 05.47.05.000000 AM
     4    19 01-FEB-16 12.53.59.000000 PM
     5     1 18-SEP-15 09.54.47.000000 AM
     5     1 13-OCT-15 05.46.39.000000 PM
     5     2 08-AUG-15 01.35.43.000000 PM
     5     5 23-MAR-15 10.39.11.000000 PM
     5     6 10-AUG-15 06.54.18.000000 AM
```

```
         5     10 17-JUN-15 03.41.04.000000 AM
         5     10 09-JAN-16 09.32.54.000000 AM
         5     11 08-JUN-15 01.36.22.000000 PM
         5     11 04-SEP-15 09.21.52.000000 AM
         5     12 28-FEB-15 07.17.53.000000 AM
         5     13 07-MAR-15 04.41.45.000000 AM
         5     13 14-MAR-15 09.47.24.000000 AM
         5     13 15-MAY-15 01.50.40.000000 AM
         5     16 15-JUN-15 03.42.38.000000 AM
         5     16 08-NOV-15 12.55.15.000000 PM
         5     17 05-OCT-15 11.57.58.000000 PM
         5     19 02-DEC-15 08.29.50.000000 PM
         6      3 25-JUN-15 11.54.35.000000 PM
         6      9 24-MAR-15 07.06.11.000000 AM
         6      9 26-APR-15 10.38.38.000000 AM
         6     10 06-MAR-15 01.29.32.000000 PM
         6     11 20-APR-15 08.44.04.000000 PM
         6     13 16-JAN-16 06.33.51.000000 PM
         6     17 04-APR-15 04.30.01.000000 AM
         6     17 02-OCT-15 10.03.40.000000 AM
         6     20 31-DEC-15 10.21.55.000000 AM
         7      1 05-NOV-15 02.48.37.000000 AM
         7      7 04-DEC-15 09.11.18.000000 PM
         7     14 08-JUN-15 04.00.28.000000 AM
         7     15 19-JUL-15 06.35.21.000000 AM
         7     15 14-DEC-15 05.05.52.000000 AM
         7     16 04-JUN-15 08.59.54.000000 AM
         7     18 25-MAR-15 07.58.23.000000 AM
         7     18 01-SEP-15 08.49.27.000000 AM
         7     20 28-APR-15 03.00.41.000000 AM
         8      3 24-OCT-15 12.54.44.000000 PM
         8      7 25-SEP-15 02.01.02.000000 PM
         8      7 09-FEB-16 07.08.21.000000 AM
         8      9 04-MAR-15 05.45.39.000000 PM
         8     13 03-MAY-15 01.21.14.000000 AM
         8     20 06-FEB-16 03.40.18.000000 AM
         9      1 01-MAY-15 11.15.47.000000 AM
         9      3 29-MAR-15 01.52.58.000000 PM
         9      6 03-JUN-15 09.56.18.000000 PM
         9     13 17-DEC-15 03.47.44.000000 AM
         9     18 06-SEP-15 10.10.10.000000 AM
         9     20 05-JAN-16 05.30.09.000000 AM
        10      7 07-DEC-15 02.32.05.000000 AM
        10      8 12-OCT-15 06.02.19.000000 PM
        10      9 26-FEB-15 03.13.41.000000 PM
        10     14 06-JUN-15 06.40.18.000000 PM
        10     16 07-FEB-16 08.49.33.000000 PM
        10     16 11-FEB-16 02.50.03.000000 PM
        10     20 24-OCT-15 03.35.38.000000 AM

100 rows selected.

WINTER342 SQL> spool off
```

Works In:

```
WINTER342 SQL> desc arjt_works_in
 Name                      Null?    Type
 ------------------------- -------- ----------------
 EMP_ID                    NOT NULL NUMBER(5)
 UNIT_ID                   NOT NULL NUMBER(5)
 START_DATE                NOT NULL DATE
 END_DATE                           DATE

WINTER342 SQL> spool off

   WINTER342 SQL> select * from arjt_works_in;

   EMP_ID UNIT_ID START_DATE   END_DATE
   ------ ------- ------------ ------------
        1       1 22-JAN-16
        2       1 19-JAN-16
       10       5 26-DEC-15    02-MAR-16
        1       4 23-DEC-15    29-FEB-16
        8       5 13-DEC-15    17-FEB-16
        7       4 24-NOV-15
        8       4 21-OCT-15    23-DEC-15
        2       3 20-OCT-15    21-DEC-15
        9       5 17-OCT-15    03-JAN-16
        7       4 22-SEP-15    04-DEC-15
        6       2 18-SEP-15
        9       5 17-SEP-15    03-DEC-15
        1       4 15-SEP-15    14-NOV-15
        4       4 01-SEP-15
       10       5 14-AUG-15    17-OCT-15
        4       5 06-AUG-15    15-OCT-15
        5       2 28-JUL-15
        5       3 09-JUL-15    26-SEP-15
        8       1 08-JUL-15    16-SEP-15
        6       5 08-JUL-15    20-SEP-15
        3       4 27-JUN-15
        5       2 25-MAY-15    28-JUL-15
        2       1 23-MAY-15    26-JUL-15
        3       5 20-APR-15    23-JUN-15
        4       1 19-APR-15    18-JUN-15
        6       1 16-APR-15    24-JUN-15
        4       4 31-MAR-15    10-JUN-15
        7       4 09-MAR-15    14-MAY-15
        2       5 24-FEB-15    06-MAY-15
        1       2 18-FEB-15    02-MAY-15

   30 rows selected.

   WINTER342 SQL> spool off
```

3.5 Example Queries in SQL

In this section, we will list the SQL implementation of each of the queries from the previous phase. We will also implement three extra queries which demonstrate SQL * Plus functionality. For each query, we will show the selection of sample data that the query returns. Note that test data was added to the already-existing sample data to ensure a minimum, testable set of data for each query.

1. List all patients who have been assessed by each of all nurses who are currently hired.

```
WINTER342 SQL> list

  select unique p.pat_id, p.first_name, p.last_name from arjt_patient
p
  where not exists ( select * from arjt_employee e
      where exists ( select * from arjt_works_in w
          where w.end_date is NULL and w.emp_id = e.emp_id
      )
      and not exists (
          select * from arjt_assessment a
          where a.emp_id = e.emp_id and a.pat_id = p.pat_id
      )
  )

WINTER342 SQL> @queries/q1

    PAT_ID FIRST_NAME                     LAST_NAME
---------- ------------------------------
------------------------------
         1 Chloe                          Price
         4 Lisa                           Plant
         5 William                        Price
         8 Dana                           Ward
        10 Rachel                         Amber

WINTER342 SQL> spool off
```

2. List all nurses who have been assigned to least 2 patients in the same time period.

```
WINTER342 SQL> list

  select unique e.emp_id, e.first_name, e.last_name
  from arjt_employee e
  where exists ( select * from arjt_assigned_to a1, arjt_assigned_to
a2
      where a1.emp_id = e.emp_id and a2.emp_id = e.emp_id
      and a1.pat_id != a2.pat_id and a1.start_date >= a2.start_date
      and a1.start_date <= a2.end_date
  )

WINTER342 SQL> @queries/q2

    EMP_ID FIRST_NAME                     LAST_NAME
---------- ------------------------------
------------------------------
         1 Albert                         Perry
         5 Jose                           Day
         6 Stephanie                      Johnson
         7 William                        Price

WINTER342 SQL> spool off
```

3. List the medications that all currently admitted patients have in common.

```
WINTER342 SQL> list

  select unique m.med_id, m.med_name, m.med_purp
  from arjt_medication m
  where not exists ( select * from arjt_patient p
      where exists ( select * from arjt_patient_admitted pa
          where pa.pat_id = p.pat_id and pa.dis_date is null
      )
      and not exists ( select * from arjt_prescription rx
          where rx.start_date <= sysdate and rx.end_date >= sysdate
          and rx.med_id = m.med_id and rx.pat_id = p.pat_id
      )
  )

WINTER342 SQL> @queries/q3

    MED_ID MED_NAME                       MED_PURP
---------- ------------------------------
------------------------------
         4 Flurazepam Hydrochloride       Laxative
         6 Loratadine                     Antibiotic
         9 Escitalopram Oxalate           Blood thinner

WINTER342 SQL> spool off
```

4. List all patients who have been admitted to the hospital exactly once.

```
WINTER342 SQL> list

  select unique p.pat_id, p.first_name, p.last_name
  from arjt_patient p
  where exists ( select * from arjt_patient_admitted pa
      where pa.pat_id = p.pat_id and
      not exists ( select * from arjt_patient_admitted pa2
          where pa2.pat_id = p.pat_id and pa.adm_date != pa2.adm_date
      )
  )

WINTER342 SQL> @queries/q4

    PAT_ID FIRST_NAME                     LAST_NAME
---------- ------------------------------
------------------------------
         2 Max                            Caulfield
         3 Kate                           Marsh
        14 Luke                           Parker

WINTER342 SQL> spool off
```

5. List all patients who have received special need orders for all of the same activities as John Doe ordered for Chelsea Doe.

```
WINTER342 SQL> list

  select p.pat_id, p.first_name, p.last_name
  from arjt_patient p
  where not exists ( select * from arjt_patient p2, arjt_special_need
s,
      arjt_employee e
      where s.pat_id = p2.pat_id and s.emp_id = e.emp_id
      and p2.first_name = 'Max' and p2.last_name = 'Caulfield'
      and e.first_name = 'David' and e.last_name = 'Madson'
      and not exists (select * from arjt_special_need s2
          where s2.act_id = s.act_id and p.pat_id = s2.pat_id
      )
  )

WINTER342 SQL> @queries/q5

    PAT_ID FIRST_NAME                     LAST_NAME
---------- ------------------------------
------------------------------
         2 Max                            Caulfield
         3 Kate                           Marsh
         4 Lisa                           Plant

WINTER342 SQL> spool off
```

6. List all of the patients who have been assessed today by at least one of the nurses who assessed John Doe on 1/27/15.

```
WINTER342 SQL> list

  select unique p.pat_id, p.first_name, p.last_name
  from arjt_patient p
  where exists ( select * from arjt_employee e, arjt_assessment a1,
      arjt_patient p1
      where a1.emp_id = e.emp_id and a1.pat_id = p1.pat_id
      and p1.first_name = 'Chloe' and p1.last_name = 'Price'
      and a1.dat = to_timestamp('5/22/2015 21:42:59', 'MM/DD/YYYY
hh24:mi:ss')
      and exists ( select * from arjt_assessment a2
          where a2.emp_id = e.emp_id and a2.pat_id = p.pat_id
          and a2.dat = to_timestamp('2/21/2016 1:35:14', 'MM/DD/YYYY
hh24:mi:ss')
      )
  )

WINTER342 SQL> @queries/q6

    PAT_ID FIRST_NAME                     LAST_NAME
---------- ------------------------------
------------------------------
         6 Warren                         Graham
         2 Max                            Caulfield
         8 Dana                           Ward
        10 Rachel                         Amber

WINTER342 SQL> spool off
```

7. List the nurses who most recently administered one of Chelsea Doe's prescriptions.

```
WINTER342 SQL> list

  select unique e.emp_id, e.first_name, e.last_name
  from arjt_employee e
  where exists ( select * from arjt_patient p, arjt_prescription rx,
      arjt_administers a
      where rx.pat_id = p.pat_id and a.rx_id = rx.rx_id
      and p.first_name = 'Chloe' and p.last_name = 'Price'
      and a.emp_id = e.emp_id
      and not exists ( select * from arjt_patient p2,
arjt_prescription rx2,
          arjt_administers a2
          where rx2.pat_id = p2.pat_id and a2.rx_id = rx2.rx_id and
          p2.first_name = 'Chloe' and p2.last_name = 'Price' and
          a2.dat > a.dat
      )
  )

WINTER342 SQL> @queries/q7

    EMP_ID FIRST_NAME                     LAST_NAME
---------- ------------------------------
------------------------------
         6 Stephanie                      Johnson

WINTER342 SQL> spool off
```

8. List nurses who have performed each of all activities in the database.

```
WINTER342 SQL> list

  select unique e.emp_id, e.first_name, e.last_name
  from arjt_employee e
  where not exists ( select * from arjt_activity a
      where not exists ( select * from arjt_special_need s,
arjt_performs p
          where s.act_id = a.act_id and p.sn_id = s.sn_id and p.emp_id
= e.emp_id
      )
  )

WINTER342 SQL> @queries/q8

    EMP_ID FIRST_NAME                     LAST_NAME
---------- ------------------------------
------------------------------
         1 Albert                         Perry
         2 Arthur                         Moore
         5 Jose                           Day

WINTER342 SQL> spool off
```

9. List all discharged patients who stayed in the same room as John Doe.

```
WINTER342 SQL> list

  select unique p.first_name, p.last_name
  from arjt_patient p
  where exists ( select * from arjt_patient p1, arjt_placed pl1
      where pl1.pat_id = p1.pat_id
      and p1.first_name = 'Chloe'
      and p1.last_name = 'Price'
      and exists ( select * from arjt_placed pl2
          where pl2.pat_id = p.pat_id and pl1.room_id = pl2.room_id
          )
      )
      and not exists ( select * from arjt_patient_admitted pa
          where pa.pat_id = p.pat_id and pa.dis_date is null
      )

WINTER342 SQL> @queries/q9

FIRST_NAME                     LAST_NAME
------------------------------ ------------------------------
Max                            Caulfield

WINTER342 SQL> spool off
```

10. List all currently-admitted patients who don't have any currently active prescriptions.

```
WINTER342 SQL> list

  select unique p.pat_id, p.first_name, p.last_name
  from arjt_patient p
  where exists ( select * from arjt_patient_admitted pa
      where pa.pat_id = p.pat_id
      and not exists ( select * from arjt_prescription rx
          where rx.start_date <= sysdate and rx.end_date >= sysdate
          and rx.pat_id = p.pat_id
      )
  )

WINTER342 SQL> @queries/q10

    PAT_ID FIRST_NAME                     LAST_NAME
---------- ------------------------------
------------------------------
         8 Dana                          Ward
        11 Frank                         Bowers

WINTER342 SQL> spool off
```

Additional Queries

In addition to the queries designed in Phase II, we have included three other queries that demonstrate Oracle and SQL*Plus functionality.

11. List all the medications and the number of times each has been prescribed.

```
WINTER342 SQL> list

  select med_name "Medication", count(*) "Number of Rx"
  from arjt_medication natural join arjt_prescription
  group by med_name
  order by count(*)

WINTER342 SQL> @queries/q11

Medication                     Number of Rx
------------------------------ ------------
Hydrocodon                                1
Ciprofloxacin                             1
Spironolactone                            2
Lisinopril                                2
Acetaminophen                             3
Cytarabine                                3
Escitalopram Oxalate                     10
Loratadine                               10
Triclosan                                15
Flurazepam Hydrochloride                 17

10 rows selected.

WINTER342 SQL> spool off
```

This query uses the *group by* clause and the *count* aggregate function to show the number of prescriptions associated with each medication. The *group by* clause combines tuples that have the same value for a set of attributes (or attribute expressions) into one tuple. It requires an aggregate function (like “count”), runs the aggregate function over attribute expressions from the grouped tuples, and returns the result as one of the columns (in this case, “Number of Rx”).

12. List all the activities and all the times they have been performed if they have been performed less than three times.

```
WINTER342 SQL> list

  select act_name "Activity", count(*) "Number of SN"
  from arjt_activity natural join arjt_special_need
  group by act_name
  having count(*) < 3

WINTER342 SQL> @queries/q12

Activity                       Number of SN
------------------------------ ------------
burn care                                 1
bathing                                   2
physical therapy                          2
bed turn                                  1
showering                                 1
catheter care                             1

6 rows selected.

WINTER342 SQL> spool off
```

This query uses the *group by* clause combined with the *having* clause. The *having* clause filters the results of the *group by* clause by evaluating conditions over the aggregate function. In this case, only the tuples were the count is less than 3 are selected. This query also uses the *natural join* operation instead of the Cartesian product (notated by a "," in queries 1-10). *Natural join* automatically equates attribute expressions that have the same name and removes duplicate columns.

13. List all employees and their prescriptions they have ordered (if they are doctors).

```
WINTER342 SQL> list

  create or replace view arjt_doctor_prescribes as
  select first_name, last_name, med_name, dat
  from arjt_employee e left outer join arjt_prescription rx on
  rx.emp_id = e.emp_id
      left outer join arjt_medication m on m.med_id = rx.med_id

WINTER342 SQL> set pagesize 10;
WINTER342 SQL> break on med_name;
WINTER342 SQL> select * from arjt_doctor_prescribes;

View created.

FIRST_NAME                     LAST_NAME                      MED_NAME                       DAT
------------------------------ ------------------------------ ------------------------------ ---------
Samuel                         Taylor                         Triclosan                      10-MAY-15
Harold                         Roberts                        Acetaminophen                  18-AUG-14
Jose                           Hall                                                          11-FEB-14
Samuel                         Taylor                                                        25-JUN-14
Ruth                           Hamilton                       Cytarabine                     22-JAN-15
Harold                         Roberts                                                       20-APR-15
David                          Madson                                                        21-AUG-14

FIRST_NAME                     LAST_NAME                      MED_NAME                       DAT
------------------------------ ------------------------------ ------------------------------ ---------
Samuel                         Taylor                         Flurazepam Hydrochloride       10-MAY-15
Jose                           Hall                                                          20-APR-15
Ruth                           Hamilton                                                      25-JUL-14
Jose                           Hall                                                          23-JAN-15
Ruth                           Hamilton                                                      27-DEC-14
Samuel                         Taylor                                                        10-MAY-15
Harold                         Roberts                                                       04-JUL-14

FIRST_NAME                     LAST_NAME                      MED_NAME                       DAT
------------------------------ ------------------------------ ------------------------------ ---------
Jose                           Hall                           Flurazepam Hydrochloride       01-FEB-16
Jose                           Hall                                                          19-AUG-14
Jose                           Hall                                                          07-JUL-15
Jose                           Hall                           Lisinopril                     04-JUN-14
Ruth                           Hamilton                                                      14-OCT-14
David                          Madson                                                        03-AUG-15
Samuel                         Taylor                         Loratadine                     10-MAY-15

FIRST_NAME                     LAST_NAME                      MED_NAME                       DAT
------------------------------ ------------------------------ ------------------------------ ---------
Samuel                         Taylor                         Loratadine                     10-MAY-15
Ruth                           Hamilton                       Ciprofloxacin                  14-OCT-14
David                          Madson                         Hydrocodon                     13-JAN-15
Samuel                         Taylor                         Escitalopram Oxalate           10-MAY-15
Harold                         Roberts                        Spironolactone                 04-JUL-14
Jose                           Hall                                                          23-SEP-14
Samuel                         Taylor                                                        25-JUN-14

FIRST_NAME                     LAST_NAME                      MED_NAME                       DAT
------------------------------ ------------------------------ ------------------------------ ---------
Stephanie                      Johnson
Arthur                         Moore
Albert                         Perry
Joyce                          Price
Deborah                        Gordon
William                        Price
Mark                           Jefferson

FIRST_NAME                     LAST_NAME                      MED_NAME                       DAT
------------------------------ ------------------------------ ------------------------------ ---------
Ruby                           Reid
Ray                            Wells
Jose                           Day

38 rows selected.

WINTER342 SQL> spool off
```

This query demonstrates several SQL * Plus functionalities. It uses the *create or replace view* command to save a query as a "virtual table," which can be queried from like a physical table. It also uses SQL * Plus commands to control the formatting of output; "break on" prevents duplicate attribute values from being displayed in a continuous sequence, and "set pagesize" determines how many records are displayed before a page break.

The query itself uses the *left outer join* operation, which displays all tuples on the left side of a join regardless of whether there are any associated tuples on the right side of the join (in this case, the right-side values are displayed as empty or "NULL"). By outer left joining *ajrt_employee* with *arjt_prescription*, all employees are displayed - including nurses - even though nurses do not order prescriptions. Variations of the *outer join* operation include *left outer join, right outer join,* and *outer join*.

3.6 Data Loader

It is important to demonstrate how large amounts of sample data is loaded into the physical implementation of the database. Methods include manually writing SQL commands to insert data and using software applications, which automatically generate SQL, insert scripts from data stored in files.

"Insert" SQL Statements
The simplest way to insert data into Oracle DBMS tables uses the "insert into" SQL statement provided by Oracle. The statement has two variations:

1. insert into **<table name>** (**<column name 1>**, …, **<column name n>** values (**<expression 1>**, … **<expression n>**)
2. insert into **<table name> <select query>**

The first variation lets you specify value expressions for each column in the table when inserting a record. The second variation lets you use the result of a query as the column values.

Since a command has to be written for each of the records that need to be inserted, this method is not desirable for loading large amounts of sample data into the database. The following methods generate and run SQL commands from files, which contain the data that needs to be inserted. As a result, these methods are far faster.

Oracle SQL Developer
SQL Developer is a free software application provided by Oracle that allows users to develop and manage Oracle databases. Using a graphical user interface, users view all the tables in the database, and can easily import data into them with CSV files. SQL Developer can also create an insert script with SQL commands should the user choose to insert the data manually. Oracle SQL Developer was the method we used to load our database with records.

Data Loader
Dr. Huaqing Wang has developed a software application, which uses a command line interface to insert data into database tables from a text file. The user specifies the name of the database, the password, and the text file to be used in the command line. The text file must follow a specific format, which specifies the data and the table into which the tuple should be inserted. However, the user can specify which character is used as a delimiter to separate columns through the command line (e.g. "," or "|"). Based on the information in the text file, "insert into" SQL statements are generated and run. This speeds up the process of inserting data because the user does not have to manually type every component of the "insert into" command.

4. Oracle Database Management System PL/SQL Components

In the previous phase of development, we showed how a physical database could be constructed with Oracle DBMS. We also showed some basic operations that could be performed on the records in the database. However, adhering to integrity constraints and business rules when manipulating a database often requires that more complex operations be defined.

In this phase, we will explore how to implement complex database operations using Oracle's procedural extension of the SQL language: PL/SQL. First, we will explain the purpose of PL/SQL as well as its benefits. After this, we will we will explore some of the features provided by PL/SQL and the syntax for utilizing each. Then, we will list and explain sample PL/SQL operations for our Oracle database. Finally, we will explore extensions of SQL offered by other popular commercial DBMS and compare them to PL/SQL.

4.1 Oracle PL/SQL

PL/SQL is Oracle's procedural language extension of SQL. It allows users to define the order in which SQL statements are executed using *flow control structures* like conditional statements and loops. PL/SQL is used to build stored procedures and functions. Stored procedures and functions are precompiled blocks of PL/SQL code that can be run at any time.

There are several advantages to using stored procedures in a database application instead of manually writing PL/SQL blocks and sending them to the server. First, stored procedures are precompiled, meaning that the PL/SQL code does not need to be compiled every time it is run; this saves time during execution. Also, stored procedures are reusable; they condense complex operations into single functions that can be repeatedly used by different developers and users. Finally, stored procedures hide complex functionality from users, making code-writing easier and safer.

4.1.1 Program Structure and Control Statements

PL/SQL is divided into collections of statements called *blocks*. Each block has its own local variables and scope. Blocks can be unnamed (or anonymous) or named. Named blocks can be procedures or functions. The syntax for an anonymous block is described below.

Syntax for anonymous block:

```
declare <variables> begin <PL/SQL statements> end;
```

PL/SQL offers *flow control structures* that control the order in which statements are executed. These include conditional statements ("if" statements) and different types of loops (*for, while*, and basic loops). The syntax for these are described below.

Syntax for conditional statements:

```
if <condition>  then
      <PL/SQL statements>
elseif <conditon> then
      <PL/SQL statements>
else
      <PL/SQL statements>
end if;
```

Syntax for loops:

```
loop
      <PL/SQL statements>
end loop;

for <counter variable> in <range> loop
      <PL/SQL statements>
end loop;

while <condition> loop
      <PL/SQL statements>
end loop;
```

4.1.2 Stored Procedures

A stored procedure is a group of SQL statements that perform a particular task. It encapsulates a set of operations or queries to execute on a database. Unlike stored functions, stored procedures do not return values. Sometimes, procedures can cause triggers to execute when a particular event occur. One advantage to using stored procedures instead of just SQL statements is that it eliminates the possibility of SQL injections. When passing parameters to a stored procedure, that procedure checks the type of that parameter to make sure that it matches the type of that stored procedure.

Syntax of a Stored Procedure:

```
create [or replace] procedure <procedure name> (
      <parameter list>
)
as
      <declarations>
begin
      <executable section>
end;
```

4.1.3 Stored Functions

A stored function is almost identical to a stored procedure, except that it has a return value. Instead of modifying data in the database, functions are meant to calculate and return a value.

Syntax of a Stored Procedure:

```
create [or replace] procedure <procedure name> (
      <parameter list>
)
return <return type>
as
      <declarations>
begin
      <executable section>
end;
```

4.1.4 Packages

Packages are Oracle Schema objects which combine procedures, functions, types, and other objects into one unit. All the objects in a package have their own namespace, so packages can be used to avoid name conflicts. Packages contain a "header" section that contains prototypes for the procedures and functions, as well as a "body" section that contains the definitions of the procedures and functions.

Syntax of a Package:

```
create or replace package <package name> as
      <function, procedure, object prototypes>
end;

create or replace package body <package name> as
      <function, procedure, object definitions>
end;
```

4.1.5 Triggers

A trigger is a stored PL/SQL procedure associated with a table, view, schema, or the database itself. A trigger will automatically execute when a certain event takes place. Triggers can be executed before, after, or instead of an event. One example where a trigger is needed is a cascade delete operation. When deleting a tuple, other tuples that reference that tuple must first be deleted. This requires what is called a cascade operation. Some cascade operations may be implemented to delete two or more levels of tuples.

Syntax of a Trigger:

```
create or replace trigger <trigger name>
<before, after, instead of> <event name> on <table name>
for each row
begin
      <PL/SQL statements>
end;
```

4.2 Oracle PL/SQL Subprogram Examples

In this section, we will list and define a package, three procedures/functions, and three triggers for our database. The package groups all of the procedures and functions together into one unit. The procedures include a procedure to insert a patient, a procedure to delete a patient, and a function to return the average heart rate for patients. The three triggers we have implemented are the cascade delete trigger, update trigger, and instead of trigger for updating a view.

Package Definition:
A package groups functions, procedures, type definitions, and other Oracle Schema objects into one unit. Members of a package are in a separate namespace than the rest of the objects in the database, so using packages can help avoid name conflicts. A package has a "header," which includes prototypes for all of the procedures and functions, as well as a "body," which defines and implements all of the procedures and functions. Below, the header for the package *arjt_pkg* is listed. *arjt_pkg* contains the *insert_patient* and *delete_patient* procedures, as well as the *average_heart_rate* function. The full definition of each procedure and each function will be listed and explained in following subsections.

```
WINTER342 SQL> @crt/procedure/arjt_pkg_header

Package created.

WINTER342 SQL> list
  1  create or replace package arjt_pkg as
  2
  3  procedure delete_patient (
  4      pid in arjt_patient.pat_id%type
  5  );
  6
  7  procedure insert_patient (
  8      ssn in arjt_patient.ssn%type,
  9      fn in arjt_patient.first_name%type,
 10      mn in arjt_patient.mid_name%type,
 11      ln in arjt_patient.last_name%type,
 12      s in arjt_patient.street%type,
 13      z in arjt_patient.zip%type,
 14      c in arjt_patient.city%type,
 15      st in arjt_patient.state%type,
 16      p in arjt_patient.phone%type,
 17      d in arjt_patient.dob%type,
 18      g in arjt_patient.gender%type,
 19      i in arjt_patient.ins_stat%type,
 20      l in arjt_patient.language%type
 21  );
 22
 23  function average_heart_rate
```

```
 24  (
 25      n number default 1
 26  )
 27  return number;
 28
 29* end arjt_pkg;
WINTER342 SQL> @crt/procedure/arjt_pkg_body

Package body created.

WINTER342 SQL> list
  1  create or replace package body arjt_pkg as
  2
  ...function definitions listed individually below...
 91
 92
 93* end arjt_pkg;
WINTER342 SQL> spool off
```

Insert Procedure Definition:

This procedure will insert a patient into the database using parameters. When this procedure is executed, all the attributes of *arjt_patient* except "pat_id" will be passed as parameters. The procedure will first find the max value for "pat_id" and increment it automatically, then insert the rest of the parameters into the proper columns of the table.

```
WINTER342 SQL> list
      procedure arjt_insert_patient (
            ssn in arjt_patient.ssn%type,
            fn in arjt_patient.first_name%type,
            mn in arjt_patient.mid_name%type,
            ln in arjt_patient.last_name%type,
            s in arjt_patient.street%type,
            z in arjt_patient.zip%type,
            c in arjt_patient.city%type,
            st in arjt_patient.state%type,
            p in arjt_patient.phone%type,
            d in arjt_patient.dob%type,
            g in arjt_patient.gender%type,
            i in arjt_patient.ins_stat%type,
            l in arjt_patient.language%type
      )
      is
            next_id arjt_patient.pat_id%type;
      begin
          select max(p.pat_id) into next_id from arjt_patient p;
```

```
          next_id := next_id + 1;

          insert into arjt_patient (
            pat_id,
            ssn,
            first_name,
            mid_name,
            last_name,
            street,
            zip,
            city,
            state,
            phone,
            dob,
            gender,
            ins_stat,
            language
          ) values (next_id, ssn, trim(fn), trim(mn), trim(ln), trim(s), z,
trim(c), trim(st), p, d, trim(g), trim(i), trim(l));

            exception
                  when others then
                        rollback;
                        dbms_output.put_line(sqlcode || ', ' || sqlerrm);
                        commit;
      end;
WINTER342 SQL> spool off
```

Executing Insert Procedure:

```
WINTER342 SQL> exec arjt_pkg.insert_patient(123456789, 'JoAnn', 'M',
'Tuazon', '1234 CSUB', 93311, 'Bakersfield', 'CA', 1234567890,
to_date('12/17/1993', 'mm/dd/yyyy'), 'F', 'Medicare', 'English');

PL/SQL procedure successfully completed.
```

Insert Procedure Results:

After the procedure executes, it will automatically insert the new patient into the next row of *arjt_patient* with the next possible "pat_id".

```
WINTER342 SQL> select * from arjt_patient where first_name = 'JoAnn';

    PAT_ID        SSN FIRST_NAME MID_NAME  LAST_NAME  STREET        ZIP CITY
STATE  PHONE       DOB        G  INS_STAT  LANGUAGE
---------- ---------- ---------- --------- ---------- ---------- ------ -----
------- ------ ----------- ---------- -- --------- ---------
```

```
        16  123456789 JoAnn      M         Tuazon     1234 CSUB   93311
Bakersfield  CA     1234567890  17-DEC-93  F  Medicare  English

WINTER342 SQL> spool off
```

Delete Patient Procedure Definition:

The *arjt_delete_pateint* procedure will delete a tuple of *arjt_pateint* when executed. For this procedure, it takes one parameter, which will be the ID of the patient that will be deleted. The execution of this procedure alone will cause an error because there are other tables that use pat_id as a foreign key.

```
WINTER342 SQL> list
      create or replace procedure arjt_delete_patient(
          pid in arjt_patient.pat_id%type
      )
      is
      begin
          delete from arjt_patient p
          where p.pat_id = pid;
          commit;
      end;
WINTER342 SQL> spool off
```

Before Delete Patient Trigger Definition:

To properly delete a patient from the database, all the tables that inclue pat_id as a foreign key also need to be deleted. This trigger will automatically execute when the *arjt_delete_patient* procedure is called. When this trigger executes, all the associated tuples with a pat_id foreign key will be deleted before the patient is deleted from the database.

```
WINTER342 SQL> list
      create or replace trigger arjt_delete_patient
      before delete on arjt_patient
      for each row
      begin
          delete from arjt_special_need sn
          where sn.pat_id = :old.pat_id;

          delete from arjt_prescription rx
          where rx.pat_id = :old.pat_id;

          delete from arjt_assessment asmt
          where asmt.pat_id = :old.pat_id;

          delete from arjt_patient_admitted pa
          where pa.pat_id = :old.pat_id;
```

```
        delete from arjt_assigned_to at
        where at.pat_id = :old.pat_id;

        delete from arjt_placed pl
        where pl.pat_id = :old.pat_id;

    exception
        when others then
            rollback;
            dbms_output.put_line(sqlcode || ', ' || sqlerrm);
            commit;
    end;
WINTER342 SQL> spool off
```

Before Delete Prescription Trigger Definition:

arjt_prescription has a pat_id attribute, but it is also associated with a *arjt_administers* tuple. The *arjt_administers* tuple that is associated with the patient must also be deleted. Before deleting the *arjt_prescription* tuple, this trigger will delete the *arjt_administers* tuple that matches the rx_id.

```
WINTER342 SQL> list
    create or replace trigger arjt_delete_prescription
    before delete on arjt_prescription
    for each row
    begin
        delete from arjt_administers a
        where a.rx_id = :old.rx_id;

    exception
        when others then
            rollback;
            dbms_output.put_line(sqlcode || ', ' || sqlerrm);
            commit;
    end;
WINTER342 SQL> spool off
```

Before Delete Special Need Trigger Definition:

arjt_special_need has a pat_id attribute, but it is also associated with a *arjt_performs* tuple. The *arjt_performs* tuple that is associated with the patient must also be deleted. Before deleting the *arjt_special_need* tuple, this trigger will delete the *arjt_performs* tuple that matches the sn_id.

```
WINTER342 SQL> list
    create or replace trigger arjt_delete_special_need
    before delete on arjt_special_need
```

```
    for each row
    begin
        delete from arjt_performs pr
        where pr.sn_id = :old.sn_id;

    exception
        when others then
            rollback;
            dbms_output.put_line(sqlcode || ', ' || sqlerrm);
            commit;
    end;
WINTER342 SQL> spool off
```

Executing Delete Procedure:

```
WINTER342 SQL> exec arjt_pkg.delete_patient(13);

PL/SQL procedure successfully completed.
```

Delete Procedure Results:

After executing and passing "13" into *arjt_pkg.delete_pateint*, the patient with a value of "13" for the "pat_id" attribute will be deleted, as well as all tables associated with that patient. When the procedure is called, it will execute the appropriate triggers, which will successfully cascade delete throughout the database.

```
WINTER342 SQL> select pat_id, first_name, last_name from arjt_patient;

    PAT_ID FIRST_NAME                     LAST_NAME
---------- ------------------------------ ------------------------------
         1 Chloe                          Price
         2 Max                            Caulfield
         3 Kate                           Marsh
         4 Lisa                           Plant
         5 William                        Price
         6 Warren                         Graham
         7 Victoria                       Chase
         8 Dana                           Ward
         9 Alyssa                         Anderson
        10 Rachel                         Amber
        11 Frank                          Bowers
        12 Stella                         Hill
        14 Luke                           Parker
        15 Daniel                         DaCosta

14 rows selected.

WINTER342 SQL> spool off
```

Average Function Result:

The average function takes a number parameter *n* and returns the average heart rate of the top *n* tuples with the highest heart rates. It uses the *order by* clause and the *rownum* variable to retrieve the top *n tuples*, and it uses the *average* aggregate function to find the average of the heart rate values.

```
WINTER342 SQL> select arjt_pkg.average_heart_rate(10) from dual;

ARJT_PKG.AVERAGE_HEART_RATE(10)
-------------------------------
                          174.7

WINTER342 SQL> spool off
```

Instead Of Trigger Definition:

The *instead of* trigger can be used to control update operations on views that join two or more base tables. When an update operation is executed, the trigger ensures that the base tables are updated instead of the view. In this case, the *instead of* trigger handles updates on a view that joins the *arjt_prescription* and *arjt_medication* tables. The *arjt_prescription* attributes are updated, and the *arjt_medication* attributes update an existing medication or are inserted into a new medication (depending on the value of med_id).

```
WINTER342 SQL> @insteadof

Trigger created.

WINTER342 SQL> list
  1  create or replace trigger arjt_prescription_info_update
  2  instead of update on arjt_prescription_info
  3  for each row
  4  declare
  5      row_cnt number;
  6  begin
  7      /* Find whether med_id references an existing med tuple */
  8      select count(*) into row_cnt from arjt_medication
  9      where med_id = :new.med_id;
 10
 11      if row_cnt = 0 then
 12          /* If med tuple does not exist, create a new one */
 13          insert into arjt_medication (med_id, med_name, med_purp)
 14          values (:new.med_id, :new.med_name, :new.med_purp);
 15      else
 16          /* else update existing med tuple */
 17          update arjt_medication mu set mu.med_name = :new.med_name,
mu.med_purp = :new.med_purp
```

```
 18          where mu.med_id = :new.med_id;
 19      end if;
 20
 21      /* update prescription with attributes from prescription table */
 22      update arjt_prescription rx
 23      set rx.med_id = :new.med_id, rx.dos = :new.dos, rx.freq = :new.freq
 24      where rx.rx_id = :old.rx_id;
 25
 26  exception
 27      when others then
 28          rollback;
 29          dbms_output.put_line(sqlcode || ', ' || sqlerrm);
 30          commit;
 31* end;
WINTER342 SQL> spool off
```

Instead Of Trigger Result:

When an update operation is performed on arjt_prescription_info, the arjt_prescription and arjt_medication attributes are updated. The prescription with an “rx_id” of “1” is updated with the no dosage and frequency information. The medication with a “med_id” of “3” is updated with the new name and purpose information.

```
WINTER342 SQL> update arjt_prescription_info set med_id = 3, med_name =
'Advil', med_purp = 'Painkiller', freq = 'Once daily' where rx_id = 1;

1 row updated.

WINTER342 SQL> select * from arjt_medication where med_id = 3;

    MED_ID MED_NAME                        MED_PURP
---------- ------------------------------ ------------------------------
         3 Advil                          Painkiller

WINTER342 SQL> select * from arjt_prescription where rx_id = 1;


     RX_ID     EMP_ID     PAT_ID     MED_ID DOS
---------- ---------- ---------- ---------- --------------------------------
-----------------
FREQ
START_DAT END_DATE
--------------------------------------------------------------------------------
---------------------- --------- ---------
DAT
---------
         1         12          1          3 113 mg
```

```
Once daily
22-AUG-14 27-SEP-14
21-AUG-14


WINTER342 SQL> spool off
```

Update Trigger Definition:

This trigger ensures that if the primary key value of a tuple in arjt_activity is changed, the value is changed for the foreign key attributes of all tuples that reference arjt_activity (in this case, ajrt_special_need is the only table).

```
WINTER342 SQL> @before_update_activity

Trigger created.

WINTER342 SQL> list
  1  create or replace trigger arjt_update_activity
  2  before update on arjt_activity
  3  for each row
  4  begin
  5
  6      update arjt_special_need sn
  7      set sn.act_id = :new.act_id
  8      where sn.act_id = :old.act_id;
  9
 10  exception
 11      when others then
 12          rollback;
 13          dbms_output.put_line(sqlcode || ', ' || sqlerrm);
 14          commit;
 15* end;
WINTER342 SQL> spool off
```

Update Trigger Result:

The value of “act_id” for a tuple in *arjt_activity* is changed from “1” to “99.” Before the update operation is committed, the trigger changes the values of “act_id” for tuples in *arjt_special_need* from “1” to “99.”

```
WINTER342 SQL> update arjt_activity set act_id = 99 where act_id = 1;

1 row updated.

WINTER342 SQL> select * from arjt_special_need where act_id = 1;

no rows selected

WINTER342 SQL> select * from arjt_special_need where act_id = 99;

     SN_ID     EMP_ID     PAT_ID     ACT_ID
---------- ---------- ---------- ----------
FREQ
START_DAT END_DATE
--------------------------------------------------------------------------------
---------------------- --------- ---------
DAT
---------
        12         12         11         99
Twice daily
06-DEC-15 31-DEC-15
06-DEC-15


WINTER342 SQL> spool off
```

4.3 PL/SQL Like Tools (Oracle, Microsoft SQL Server, and MySQL)

The physical database and procedures for this project were implemented using Oracle PL/SQL. However, there are other popular commercial DBMS software that each offer unique stored procedure functionality. In this section, Oracle PL/SQL, Microsoft SQL Server T-SQL (Transact-SQL), and MySQL will be explored and compared in terms of stored procedure functionality and syntax.

Microsoft SQL Server: T-SQL

Comparison to other commercial DBMS languages:
T-SQL for Microsoft SQL Server offers some unique functionality compared to other DBMS languages. There are options available for encrypting the text of the procedure as well as restricting user permissions (using the *with* clause). Also, T-SQL allows for multiple *try-catch* blocks in a procedure, unlike PL/SQL and MySQL, which each use a separate section for handling exceptions. Unlike in MySQL, T-SQL functions can easily return both tables and scalar values (it is possible to do this in Oracle using a *pipelined table function, but not as easily*). T-SQL has a unique method for passing and using parameters; the '@' character must precede all parameters. This is unlike MySQL and PL/SQL. T-SQL has advantages over the other DBMS languages, but it also lacks some functionality. Unlike Oracle, T-SQL does not offer *for* loops; only basic loops and *while* loops are available. Also, T-SQL does not allow procedures to be grouped into packages.

Syntax for creating a procedure/function:

```
CREATE { PROC | PROCEDURE } [schema_name.] procedure_name [ ; number ]
    [ { @parameter [ type_schema_name. ] data_type }
        [ VARYING ] [ = default ] [ OUT | OUTPUT | [READONLY]
    ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
[ FOR REPLICATION ]
AS { [ BEGIN ] sql_statement [;] [ ...n ] [ END ] }
[;]

<procedure_option> ::=
    [ ENCRYPTION ]
    [ RECOMPILE ]
    [ EXECUTE AS Clause ]


CREATE FUNCTION [ schema_name. ] function_name

( [ { @parameter_name [ AS ][ type_schema_name. ] parameter_data_type
    [ = default ] [ READONLY ] }
    [ ,...n ]
  ]
)
RETURNS return_data_type
```

```
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    BEGIN
        function_body
        RETURN scalar_expression
    END
[ ; ]
```

Syntax for basic loop:

```
WHILE @cnt < cnt_total
BEGIN
   {...statements...}
   SET @cnt = @cnt + 1;
END;
```

MySQL

Comparison to other commercial DBMS languages:

MySQL offers similar functionality to PL/SQL and T-SQL while missing a few of the features. MySQL offers most of the basic control structures available in PL/SQL, but does not offer *for loops*; only *while* loops and basic loops are available. MySQL does not offer packages for namespace management like PL/SQL does. Parameters are passed in the same way as in PL/SQL (unlike T-SQL). It is important to note that when creating a procedure in MySQL, the *delimiter* command must be used to change the default end-line character from a semicolon (;) to "//"; otherwise, only the first line of the procedure will be stored.

Syntax for creating a procedure/function:

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body

CREATE
    [DEFINER = { user | CURRENT_USER }]
    FUNCTION sp_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body
```

Syntax for basic loop:

```
[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

Oracle: PL/SQL

Comparison to other commercial DBMS languages:
PL/SQL – Oracle's procedural SQL-based language for Oracle DBMS – was used to implement the physical database for Charity General Hospital. It provides several unique features compared to the other DBMS languages, including packages, which prevent name conflicts, and more complex control structures like the *for* loop. The parameter-passing mechanism is mostly the same as in MySQL.

Syntax for creating a procedure/function:
create or replace procedure **<procedure name>** begin **<PL/SQL statements>** end
create or replace function <**function name>** return **<return type>** begin <**PL/SQL statements>** end

Syntax for basic loops:

```
while <condition>
loop
      <statements>
end loop;

for <counter variable> in <range>
loop
      <statements>
end loop;
```

5. Graphical User Interface Implementation

In this phase, we will implement a database application with a graphical user interface (GUI). We will first list all of the user groups that may user the application, including their specific needs. Then, we will describe the Oracle DBMS features required to implement the software application, including views and stored procedures. After this, we will provide screenshots for all of the features in a component of the application designed for *one* of the user groups. Then, we will describe some of the implementation process for the application, including snippets of code. Finally, we will provide an overview of the database implementation process and important lessons learned.

5.1 Daily User Activities

When designing a front-end application for a database, it is important to remember that the application will be used by a variety of users, each of whom has different needs and expectations. Each distinct user group will therefore require a different graphical user interface that provides unique functionalities. We will now list each of the user groups for Charity General Hospital and the unique required functionalities. For this project, we will only implement the interface for one of the user groups.

5.1.1 Registered Nurse Users

Registered Nurses continuously monitor and provide services to their assigned patients. They must be aware of all of the services ordered for patients by doctors, and they must document all of their actions.

Required Functionalities:

- View assigned patients
- Record health assessments for assigned patients
- View doctor-ordered prescriptions for patients
- Record when prescriptions are administered

5.1.2 Charged/Manager Nurse Users

Charged nurses supervise registered nurses during a shift and assign each nurse to specific patients.

Required Functionalities:

- Assign nurses to patients
- View responsibilities for subordinate nurses and assess nurse's actions during a shift
- Manage employment status of subordinate nurses

5.1.3 Doctor Users

Doctors order services for each patient. Services ordered include prescribed medication and special needs ordered. Nurses carry out orders made by doctors.

Required Functionalities

- Manage prescriptions and special needs activities for patients
- View patient assessment history

5.1.4 Executive/Administrative Users

Administrator users make sure the database accurately reflects the state of the hospital. Executive users view information about hospital resource usage to make decisions about finances and human resources.

Required Functionalities

- View hospital resource usage reports
- Manage patient, rooms, units, and employee information

5.2. Relations, views, and subprograms

In order to provide the services required by the registered nurses, the database application needs to take advantage of some of the PL/SQL functionalities provided by Oracle DBMS (detailed in Phase II and Phase III of development). Functionalities include views and stored procedures. The purpose of each view and stored procedure will be listed below.

View: Nurse Full Name
This view displays the full name of all hired nurses in the format: *LastName, FirstName*. The view is useful because it hides the logic needed to format the *last_name* and *first_name* fields of the employee table, as well as the logic that filters doctors from nurses. It also condenses the information into one display field, which is required by the drop down list control.

View: **Prescription Detail**
This view joins the medication, prescription, and employee tables to show all relevant information about a prescription. It also hides unnecessary fields that registered nurses do not need to access. It is used to populate the prescription list.

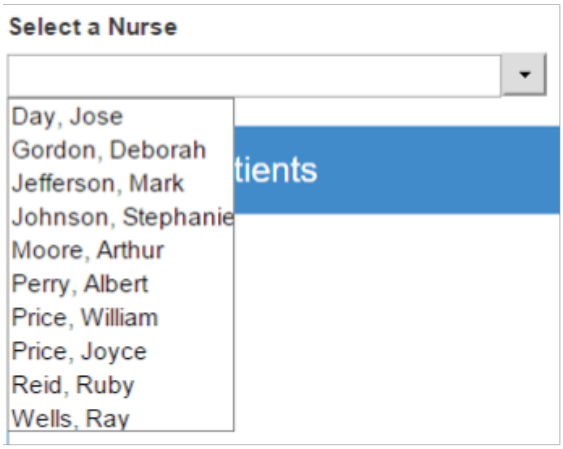
View: Prescription Summary

This view groups prescriptions by medication type and calculates the percentage of prescriptions that use each medication, given a range of dates. The view hides the logic for ensuring prescriptions are within a date range, and is required to generate the Services Summary report.

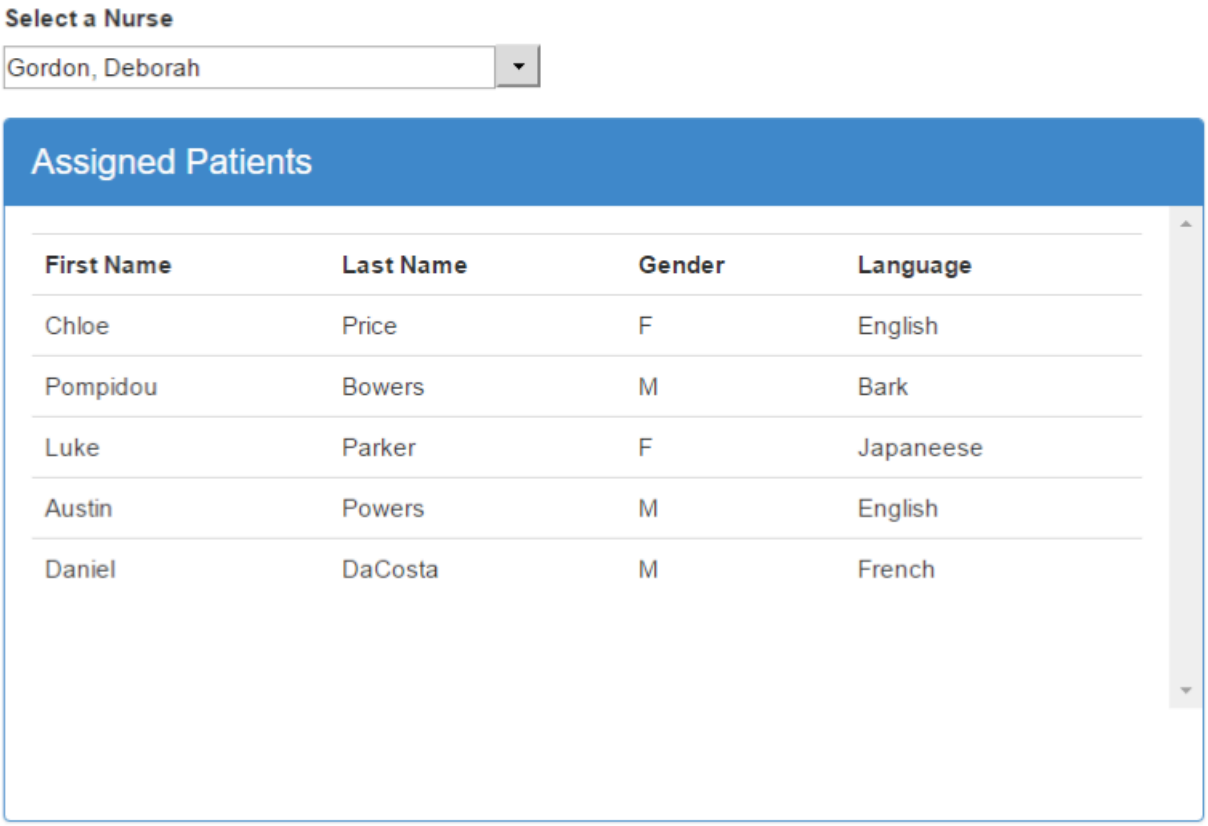
Stored Procedure: Insert Assessment

This procedure takes the parameters necessary to create an assessment and automatically calculates the next auto-increment primary key for the assessment. This is useful because it prevents developers from miscalculating primary keys and creating duplicates.

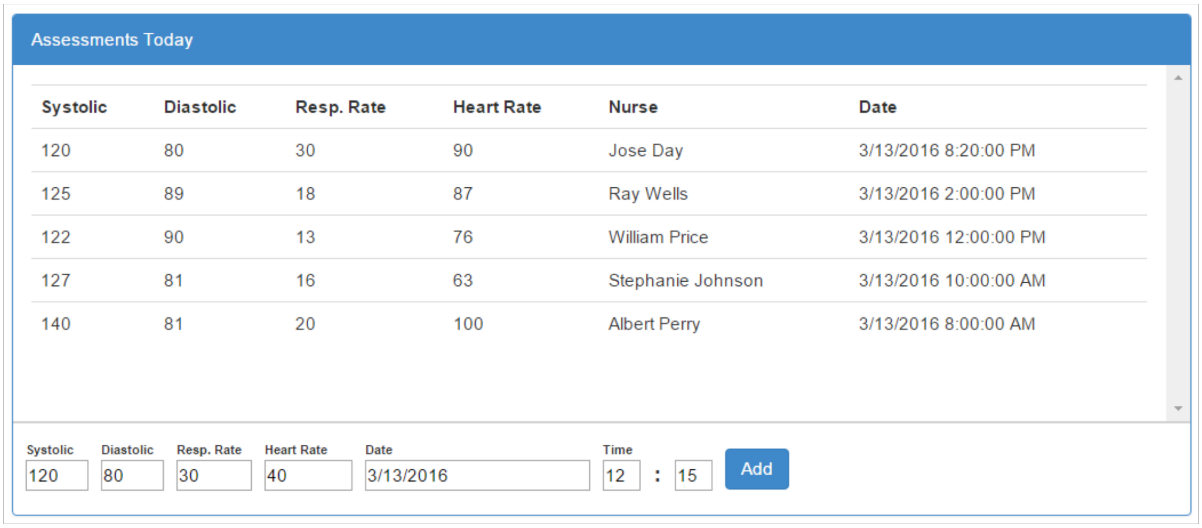
5.3 Menus and Displays



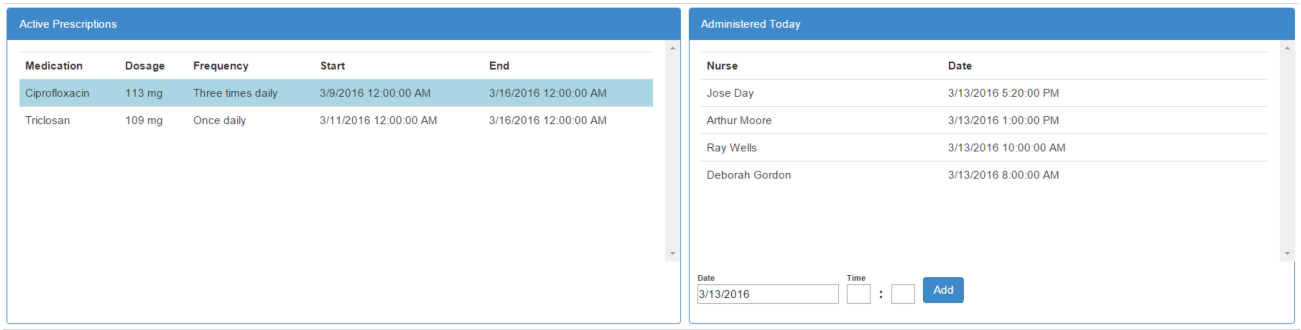
Nurses can use the dropdown list to choose their name or type their last name to automatically complete. The dropdown list displays the currently hired nurses of the hospital. When the nurse chooses their name, the patients they are assigned to that day will appear in the box below.



When a nurse is selected, the query returns the list of patients that are assigned to the nurse that day. The patient's name, gender, and language are displayed in a table. The nurse is then able to select a patient, which will then display the list of assessments of the selected patient and all of their active prescriptions.



When a patient is selected, the list of the assessments each nurse has performed that day is displayed to the right of the patient list. The nurse can enter the patient's systolic and diastolic blood pressure, respiration rate, and heart rate. The date is automatically set to today's date and they can set the time of the assessment in military time format.



When a patient is selected, their active prescriptions appear below the list of patients. The prescription box displays the name of the medication, the dosage, frequency, and the start and end date of what the doctor has prescribed for them. When a prescription is selected, the box displays the list of times that other nurses have administered that prescription for that day. After a nurse has administered a medication, the nurse may select a date, which is automatically selected to the current date, and the time it was administered.

Nurse Overview

Date: Tuesday, March 15, 2016

Employee: Gordon, Deborah

Total Patients: 5

License No.: RN 396551

Total Prescriptions: 20

Prescriptions for Assigned Patients

Patient: Price, Chloe

Medication: Triclosan **Dosage:** 109 mg **Frequency:** Once daily

Ordered by: Madson, David **Period:** 3/11/2016 12:00:00 AM - 3/16/2016 12:00:00 AM

Administered by:	Nurse License No.:	Date/Time:
Gordon, Deborah	RN 396551	3/13/2016 10:00:00 AM
Day, Jose	RN 342062	3/13/2016 1:00:00 PM

Patient: Price, Chloe

Medication: Ciprofloxacin **Dosage:** 113 mg **Frequency:** Three times daily

Ordered by: Hamilton, Ruth **Period:** 3/9/2016 12:00:00 AM - 3/16/2016 12:00:00 AM

Administered by:	Nurse License No.:	Date/Time:
Gordon, Deborah	RN 396551	3/13/2016 8:00:00 AM
Wells, Ray	RN 916003	3/13/2016 10:00:00 AM
Moore, Arthur	RN 365620	3/13/2016 1:00:00 PM
Day, Jose	RN 342062	3/13/2016 5:20:00 PM

Patient: Powers, Austin

Medication: Spironolactone **Dosage:** 56 mg **Frequency:** Twice daily

Ordered by: Roberts, Harold **Period:** 3/11/2016 12:00:00 AM - 3/15/2016 12:00:00 AM

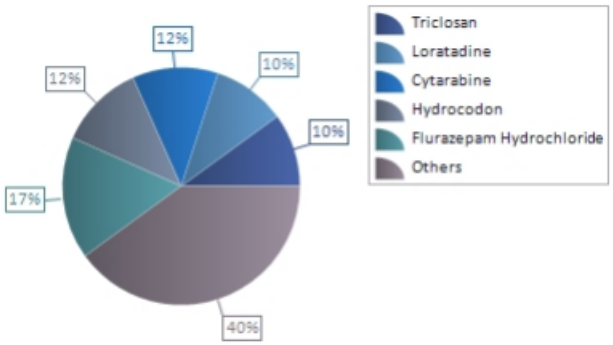
Administered by:	Nurse License No.:	Date/Time:
Wells, Ray	RN 916003	3/13/2016 10:00:00 AM
Day, Jose	RN 342062	3/13/2016 1:00:00 PM

For the first report when a nurse and patient are selected, the report dynamically shows the information of the nurse, how many total patients they are assigned to and how many prescriptions they are responsible for that day. Each prescription is displayed with the patient it corresponds with along with the dosage, frequency, and which doctor it was ordered by. This report is useful because it can check for errors if a nurse has administered a prescription more than what was listed as the frequency.

Charity General Hospital

Prescription Summary

02/14/2016 - 03/15/2016



Medication Count: 10

Total Prescriptions: 60

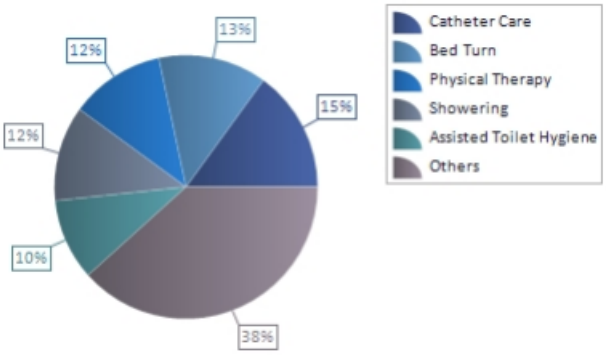
Details

Medication Name	Percentage	Number of Times Prescribed
Acetaminophen	6.67%	4
Escitalopram Oxalate	6.67%	4
Lisinopril	8.33%	5
Ciprofloxacin	8.33%	5
Triclosan	10%	6
Loratadine	10%	6
Spironolactone	10%	6
Cytarabine	11.67%	7
Hydrocodon	11.67%	7
Flurazepam Hydrochloride	16.67%	10

Charity General Hospital

Special Needs Summary

02/14/2016 - 03/15/2016



Activity Count: 10

Special Need Orders: 60

Details

Activity Name	Percentage	Number of Orders
Catheter Care	15%	9
Bed Turn	13.33%	8
Physical Therapy	11.67%	7
Showering	11.67%	7
Assisted Toilet Hygiene	10%	6
Burn Care	10%	6
Bathing	10%	6
Functional Mobility	8.33%	5
Wound Care	6.67%	4
Assisted Feeding	3.33%	2

For the second report, the nurse may choose a range of dates they wish to view for a summary of services they have provided. The first part of the report shows a summary of prescriptions with a pie chart to visualize the amount of times each medicine has been prescribed. Below the chart, is the list of the medicines and the percentage and number of times each medicine has been prescribed. This is useful because it shows which medicine is used most in the hospital so that in future orders, the hospital can order more of that specific medicine. Similar to the prescription summary, a second page lists the different types of special needs that have been performed within the date range. This information can be useful for human resources, because it indicates which activities nurses should be trained on.

5.4 Description of Code

The following section will give a brief overview of how the functionalities for the registered nurse were implemented in software.

ASP.NET Runtime Environment and C#

The application runs within the ASP.NET runtime environment, using XML files to describe the layout and components of interface pages, and C# code to handle logic. ASP.NET automatically provides threading services, and uses event handler functions to let the user manipulate pages during the various stages of their loading cycle. One example is the “Page_Load” event handler, which fires every time the client browser finishes loading a response from the server. In our Page_Load handler, the default values for UI components are set (like today’s date).

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack && !IsCallback)
    {
        AsmtDate.Text = "3/13/2016";
        AdmsDate.Text = "3/13/2016";


        SummaryStartDateText.Text = DateTime.Now.AddDays(-30).ToString(@"MM\/dd\/yyyy");
        SummaryEndDateText.Text = DateTime.Now.ToString(@"MM\/dd\/yyyy");
    }

}
```

5.4.1 Database Connection and Interaction

Connection strings (which hold information required to connect to the database) are stored in a global configuration file called web.config. The hostname and login information are stored for use throughout the application.

```
<connectionStrings>
  <add name="DefaultConnection" connectionString="Data Source=(LocalDb)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\aspnet-DatabasesFrontEnd-20160127053321.mdf;
  <add name="DelphiConnectionString" connectionString="DATA SOURCE=Delphi;PASSWORD=c3m4p2s;USER ID=WINTER342" providerName="Oracle.DataAccess.Client" />
</connectionStrings>
```

Components from the database are populated using “DataSource” objects, which specify a SQL statement, a connection string, and parameters that can be populated from other controls on the page. The Nurse Drop Down List populated using a datasource.

```
<asp:SqlDataSource ID="PatientListDataSource" runat="server" ConnectionString="<%$ ConnectionStrings:DelphiConnectionString %>" ProviderName="<%$ ConnectionStrings:De
    SelectCommand="select unique p.pat_id, p.first_name, p.last_name, p.gender, p.language from arjt_patient p where exists ( select * from arjt_employee e, arjt_assi
    OnSelecting="PatientListDataSource_Selecting">
    <SelectParameters>
        <asp:ControlParameter ControlID="NurseDDL" PropertyName="SelectedValue" Type="Int32" DefaultValue="0" Name="emp_id" />
    </SelectParameters>
</asp:SqlDataSource>
```

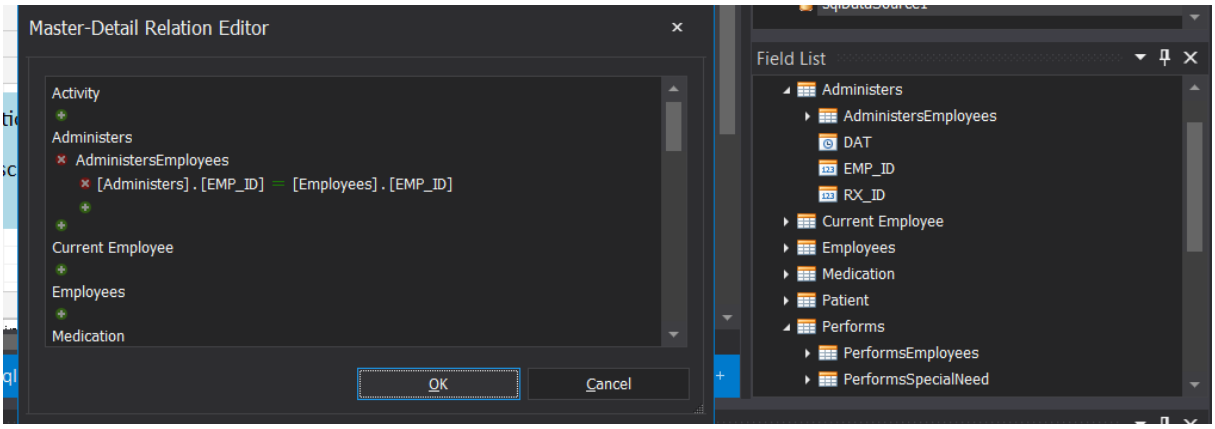
Stored procedures can be executed in C# code using the *Oracle.DataAccess* library. The *OracleCommand* class and *Oracle Connection* lets developers specify the type of command and parameters with data types. Here, the stored Insert Assessment procedure is executed in C# code. The connection is opened at the beginning of the request and closed before the request ends.

```
protected void AddAssessmentButton_Click(object sender, EventArgs e)
{
    using (OracleConnection objConn = new OracleConnection(System.Configuration.ConfigurationManager.ConnectionStrings["DelphiConnectionString"].ConnectionStr
    {
        OracleCommand objCmd = new OracleCommand();
        objCmd.Connection = objConn;
        objCmd.CommandText = "arjt_insert_assessment";
        objCmd.CommandType = System.Data.CommandType.StoredProcedure;
        objCmd.Parameters.Add("e", OracleDbType.Int32).Value = Convert.ToInt32(NurseDDL.SelectedValue);
        objCmd.Parameters.Add("p", OracleDbType.Int32).Value = Convert.ToInt32(PatientGridView.SelectedValue);
        objCmd.Parameters.Add("bs", OracleDbType.Int32).Value = Convert.ToInt32(BPSysTextBox.Text);
        objCmd.Parameters.Add("bd", OracleDbType.Int32).Value = Convert.ToInt32(BPDiaTextBox.Text);
        objCmd.Parameters.Add("rr", OracleDbType.Int32).Value = Convert.ToInt32(RespRateTextBox.Text);
        objCmd.Parameters.Add("hr", OracleDbType.Int32).Value = Convert.ToInt32(HrtRateTextBox.Text);

        DateTime dateTime = Convert.ToDateTime(AsmtDate.Text);
        dateTime = dateTime.AddHours(Convert.ToDouble(AsmtHourText.Text));
        dateTime = dateTime.AddMinutes(Convert.ToDouble(AsmtMinText.Text));

        OracleTimeStamp dat = new OracleTimeStamp(dateTime);
        objCmd.Parameters.Add("d", OracleDbType.TimeStamp).Value = dat;

        objConn.Open();
        objCmd.ExecuteNonQuery();
        objConn.Close();
    }
```

5.4.2 Reports and Report Generator

The DevExpress XtraReports feature provides a GUI interface for designing reports that generates C# files. C# code can be used to generate the pages of report and export them in pdf format.

XtraReports provides functionality for creating queries and specifying join relationships so data can automatically populate report components.



5.4.3 Major Features

The major features of the program include being able to add health assessments for selected patients given by a selected nurse. The interface also allows users to create records for administering a selected prescription. Currently, all recording processes are done by hand, and although this is required in many hospitals for legal reasons, being able to select nurses and patients rather than writing their information greatly speeds up the process.

5.4.4 Learning New Tools

Much of the learning process for ASP.NET C# and DevExpress XtraReports involves researching already-implemented examples and questions in forums and message boards. Many developers have encountered the same problems when using the framework, and as a result, answers already exist. Another useful tool is reading the documentation provided by Microsoft and DevExpress, as the documentation often contains tutorials. Also useful is exploring the many classes provided in Microsoft C# and their members. Visual Studio's *IntelliSense* feature makes exploring class members very easy.

5.5 Design and Implementation Process

Before this report concludes, it is useful to provide an overview all of the steps for implementing a database application, including lessons learned from this development process.

Requirements Collection and Analysis

This phase involves understanding the business, including its components and how they are related. It is important to perform a detailed survey of the business so its operation can be understood completely.

Conceptual Database Design

This part includes developing a detailed, visual representation of the organization's structure based on the data collected in the previous phase. It is important that multiple designs are created and the potential problems of each fully explored, as the conceptual design will form the groundwork for the physical implementation.

Logical Database Design

It is important to convert the conceptual design into one that can be used in a software implementation. The process for converting must be followed accurately and precisely, and resulting relations must be normalized to ensure the design is sound.

Physical Database Implementation

A DBMS with useful functionality must be chosen, and the logical design must be closely followed when constructing the database. The database should be loaded with plenty of data so potential problems are visible. Stored procedures and views should be design to make application development easier.

Database Application

A user interface that connects to the database must be designed for business user. Ease of use must be the key focus of design. The users should be able to manipulate and access the database without wasting any unnecessary time.

5.6 Embedded Questions

Outcome	Alex Rinaldi	JoAnn Tuazon
An ability to analyze a problem, and identify and define the computing requirements and specifications appropriate to its solution.	9	9
An ability to design, implement and evaluate a computer-based system, process, component, or program to meet desired needs, An ability to understand the analysis, design, and implementation of a computerized solution to a real-life problem.	8	8
An ability to communicate effectively with a range of audiences. An ability to write a technical document such as a software specification white paper or a user manual.	8	9
An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices.	9	8