

CLINICAL TRIALS DATABASE

CMPS 3420 SPRING 2018

BRYAN SANDERS

CESAR ALEMAN

TABLE OF CONTENTS

<u>1.1 Fact-Finding Techniques and Information Gathering....</u>	
<u>1.1.1 Introduction to Organization.....</u>	4
<u>1.1.2 Description of Fact-Finding Techniques</u>	4
<u>1.1.3 Conceptual Database</u>	4
<u>1.1.4 Entity and Relationship Sets Description</u>	4
<u>1.1.5 User Groups, Data Views, and Operations</u>	8
<u>1.2 Conceptual Database Design</u>	8
<u>1.2.1 Entity Set Description.....</u>	8
<u>1.2.2 Relationship Set Description</u>	14
<u>1.2.3 Related Entity Set</u>	17
<u>1.2.4 ER Diagram</u>	18
<u>2. Conceptual Database and Logical Database</u>	18
<u>2.1 E-R Model and Relational Model.....</u>	19
<u>2.1.1 Description of E-R Model and Relational Model.....</u>	19
<u>2.1.2 Comparison of Two Different Models</u>	19
<u>2.2 Conversion of Conceptual Database Model to Logical Database Model</u>	20
<u>2.2.1 Converting Entity Types to Relations.....</u>	20
<u>2.2.2 Converting Relationship Types to Relations</u>	21
<u>2.2.3 Database Constraints</u>	23
<u>2.3 Convert Entity Relationship Model to Relational Model</u>	24
<u>2.3.2 Sample Data of Relation</u>	33
<u>2.4 Sample Queries</u>	37
<u>2.4.1 Design of Queries</u>	37
<u>2.4.2 Relational Algebra Expressions for Queries.....</u>	37
<u>2.4.3 Tuple Relational Calculus Expressions for Queries</u>	42
<u>2.4.4 Domain Relational Calculus Expressions for Queries.....</u>	46
<u>3. SQL Server and SQL Server Management Studio</u>	48

<u>3.1.1 Normalization and Normal Forms</u>	49
<u>3.1.2 Normal Forms for Our Own Database.....</u>	51
<u>3.3 Schema Objects for SSMS.....</u>	55
<u>3.4.2 Example Queries in SQL.....</u>	67
<u>4. Oracle Database Management System PL/SQL Components.....</u>	71
<u> 4.1 Oracle PL/SQL</u>	71
<u> 4.1.1 Stored Procedures</u>	71
<u> 4.1.2 Triggers.....</u>	72
<u> 4.1.3 Putting Our Triggers and Stored Procedures into Packages</u>	72
<u> 4.2 Postgres PL/pgSQL</u>	72
<u> 4.3 Postgres/pgSQL Subprograms.....</u>	73
<u> 4.4 PL/SQL Like Tools (Oracle, Microsoft SQL Server, and MySQL).....</u>	76
<u>5. Graphical User Interface Design and Implementation</u>	80
<u> 5.1 Functionalities and User Group of the GUI Application</u>	80
<u> 5.1.1 Itemized Descriptions of GUI Application</u>	79
<u> 5.1.2 Screen Shots of Application.....</u>	80
<u> 5.2 Programming Sections.....</u>	88
<u> 5.2.1 Server-Side Programming</u>	88
<u> 5.2.2 Middle-Tire Programming.....</u>	92
<u> 5.2.3 Client-Side Programming</u>	93
<u> 5.3 Survey Questions.....</u>	94

1.1 Fact Finding Techniques and Information Gathering

To build a database for an organization that is conducting clinical trials, we first must see what is required and what is needed of the database. Section one is about pre-planning. We must gather information and make visual representation of what we want to be within our database.

1.1.1 Introduction to Organization

For us to create this database, we must first understand this business. Section 1.1 will be on what the organization does daily. In the world of test drugs and other medical advances there must always be clear and concise documentation. There are a number of important factors to keep track of which include: doctors, subjects, sites, appointments, etc. The overall goal of this organization, is to eventually find helpful drugs/equipment that will better the lives of those of are hindered by disabilities of many varieties.

1.1.2 Description of Fact-Finding Techniques

To effectively create a database, those creating must interview those will actually be interacting with the database. We gathered our information by first interviewing a group working out of the San Gregorio Hospital in Beaumont CA. Between that, some internet searching, and our intuition, we've assembled as closely as possible a system we believe can meet the needs of nearly any Clinical Trial Management group.

1.1.3 Conceptual Database

The business of Clinical Trials is massive. We will not be able to cover the entity of the whole industry, so we will be covering one fictitious company. Our ‘miniworld’ will contain a company that will conduct multiple studies, have multiple doctors and patients, multiple sites, and other key factors that will be covered in section 1.2. The overall goal is to make the process of starting, conducting, and completing a Clinical Trial as efficient as possible while retaining as much useful data as possible to report on later. Regulatory information, patient results, visit records, and any other form that needs to be completed already exists in the real world, so we will only be storing that information in the form the most common file types in our Document Info table (PDF’s, Word, Excel, Text, ect.). To remain compliant with existing laws regarding sensitive personal information, access to documents associated with a study will be limited to certain User Types.

1.1.4 Entity and Relationship Sets Description

This section will contain a basic over view of our entity sets as well as our relationship sets.

ENTITY TYPES:

- Doctor: docPID, docName, docDescription

A doctor works at a site and is assigned to a particular study. They have an ID, a name, and a description of their specializations.

- Subject: subID, subInitials, subGender, subDateOfBirth, subAge, subHeight, subWeight, subBloodType

The subjects are the most important entity to keep track of because through them we will know if a prototype has an effect or not. Due to this they contain a lot of attributes that pertain to personal and physical information.

- Site: sitPID, sitName, sitDescription, sitLocation?

The sites will be where the particular projects take place. So, they must include an ID, a name of the site, a description, and a location.

- Project: proPID, proName, proDescription

Projects will be another main focus in this company, they include an ID, name, and a detailed description.

- Study: stuPID, stuName, stType, stuEnrollmentGoal

Another critical part of the companies are the studies, which will be the main brain that gets the information from subjects, doctors, and also handles documents.

- Study_Arm: sarPID, sarName, sarDescription

The study arms will be a branch of a particular study, it will be working on a different section of the study. So, it includes, an ID, a name, and a description.

- Activity: actPID, actName, actDescription, actDuration, actCost

Activities will be tasks that the subjects using prototypes are going to be asked to perform in order to see what effects they have. They include: an ID, a name, a description, the duration, and cost.

- Visit: visPID, visName, visDescription, visStartTime, visEndTime

Visits will be primarily just to check the progress of a subject and they will include: an ID, a name, a description, and a start/end time.

- Appointments: appPID, appStartTime, appEndTime

Appointments will come about if there is an activity and/or a visit scheduled. They include: an ID, a name, and a start/end time.

- Sponsor: spoPID, spoName, spoDescription

As the name states, sponsors are the entity that provide the funding for the project. A sponsor includes: an ID, a name, a description.

- Document: docPID, docDescription, docFileName, docFileType, docBLOB, docVersion

Documents are essential to this company because that is how results and progress is kept. Documents will include: an ID, a name, a file name, file type, a BLOB, and a version of the document.

RELATIONSHIP TYPES:

Doctors <u>works at</u> site:	Appointment is <u>conducted by</u> doctor:
Cardinality: 0..1	Cardinality: 0..1
Participation: partial, total	Participation: partial, total
Site <u>Hosts</u> a project:	Appointment is <u>linked to</u> study arm:
Cardinality: 0..*	Cardinality: 0..*
Participation: partial, total	Participation: partial, total
<u>pays for</u> project:	Sponsors
Cardinality: 0..*	
Participation: partial, total	
Study <u>has</u> documents:	
Cardinality: 0..*	
Participation: partial, total	
Activity is a <u>part of</u> study arm:	
Cardinality: 0..*	
Participation: partial, total	
Visit is a <u>part of</u> a study arm:	
Cardinality: 0..*	
Participation: partial, total	
Appointment is <u>generated from</u> an activity/visit:	
Cardinality: 1..*	
Participation: partial, total	
Appointment is <u>completed by</u> a subject:	
Cardinality: 0..*	
Participation: partial, total	

1.1.5 User Groups, Data Views, and Operations

This section will contain details about the types of Users who will be interacting with our database, what kind of access they will have (permissions) and the kinds of data they will need to see. (Completed at Phase V)

1.2 Conceptual Database Design

To better grasp the scope and limitations of our data model, this section is dedicated to mapping the relations and relationships between entities using different methods. Additionally, the information below will provide the reader with specifics about integrity constraints in our database fields.

1.2.1 Entity Set Description

Below is an outline of all the entities listed in our ER Diagram, as well as their corresponding field names and integrity constraints.

User

	<i>usePID</i>	<i>useName</i>	<i>usePassword</i>	<i>useType</i>
<i>Description</i>	This ID is used to uniquely identify a User.	This is used to provide the user with a unique string of characters needed for logging in	This field stores the password created for the User. The value should be hashed & salted for added security.	This field is an integer used as a reference for the User Type table
<i>Domain/Type</i>	integer	varchar(50)	varchar(50)	integer
<i>Value Range</i>	All positive integers	String of ASCII maximum 50	String of ASCII maximum 50	All positive integers
<i>Default Value</i>	NASK	None	None	3
<i>Nullable?</i>	No	No	No	No
<i>Unique?</i>	Yes	Yes	No	No
<i>Single or Multiple Value</i>	Single	Single	Single	Single
<i>Simple or Composite</i>	Simple	Simple	Simple	Simple

- Candidate keys: usePID, useName
- Primary key: usePID
- Strong/Weak Entity: Strong
- Fields to be indexed: usePID, useName

User Type

	<i>utyPID</i>	<i>utyName</i>	<i>utyDescription</i>
<i>Description</i>	This ID is used to uniquely identify a User Type.	This is a descriptive label assigned to a User Type for clarity.	This field stores the description of the particular User Type to be referenced when needed.
<i>Domain/Type</i>	integer	varchar(50)	varchar(200)
<i>Value Range</i>	All positive integers	String of ASCII maximum 50	String of ASCII maximum 200
<i>Default Value</i>	NASK	Un-named Type	NULL
<i>Nullable?</i>	No	No	Yes
<i>Unique?</i>	Yes	Yes	No
<i>Single or Multiple Value</i>	Single	Single	Single
<i>Simple or Composite</i>	Simple	Simple	Simple

- Candidate keys: utyPID

- Primary key: utyPID
- Strong/Weak Entity: Weak
- Fields to be indexed: utyPID

Site

	<i>sitPID</i>	<i>sitName</i>	<i>sitDescription</i>
<i>Description</i>	This ID is used to uniquely identify a Site.	This is a label applied for the User's benefit to a Site	This field stores a brief description of a Site, it could contain almost anything.
<i>Domain/Type</i>	integer	varchar(50)	varchar(200)
<i>Value Range</i>	All positive integers	String of ASCII maximum 50	String of ASCII maximum 200
<i>Default Value</i>	NASK	None	NULL
<i>Nullable?</i>	No	No	Yes
<i>Unique?</i>	Yes	Yes	No
<i>Single or Multiple Value</i>	Single	Single	Single
<i>Simple or Composite</i>	Simple	Simple	Simple

- Candidate keys: sitPID, sitName
- Primary key: sitPID
- Strong/Weak Entity: Strong
- Fields to be indexed: sitPID, sitName

Sponsor

	<i>spoPID</i>	<i>spoName</i>	<i>spoDescription</i>	<i>spoPhoneNumber</i>
<i>Description</i>	This ID is used to uniquely identify a Sponsor.	This column holds the unique company name of the sponsor participating in Studies.	This field stores a brief description of the sponsor (to be used if needed)	This field holds the 9 digit primary phone number of a Sponsor.
<i>Domain/Type</i>	integer	varchar(50)	varchar(200)	integer
<i>Value Range</i>	All positive integers	String of ASCII maximum 50	String of ASCII maximum 200	All 9-digit positive integers
<i>Default Value</i>	NASK	None	NULL	NULL
<i>Nullable?</i>	No	No	Yes	Yes
<i>Unique?</i>	Yes	Yes	No	No
<i>Single or Multiple Value</i>	Single	Single	Single	Single
<i>Simple or Composite</i>	Simple	Simple	Simple	Simple

- Candidate keys: spoPID, spoName
- Primary key: spoPID
- Strong/Weak Entity: Strong
- Fields to be indexed: spoPID, spoName

Doctor

	<i>docPID</i>	<i>docName</i>	<i>docSpecialization</i>	<i>docPhoneNumber</i>
<i>Description</i>	This ID is used to uniquely identify a Doctor.	This contains a Doctor's professional title, including their credential abbreviations	A Doctor's main area of interest within a Study will be stored here.	This field holds the 9 digit primary phone number of a Doctor.
<i>Domain/Type</i>	integer	varchar(75)	varchar(50)	integer
<i>Value Range</i>	All positive integers	String of ASCII maximum 75	String of ASCII maximum 50	All 9-digit positive integers
<i>Default Value</i>	NASK	None	NULL	Null
<i>Nullable?</i>	No	No	Yes	Yes
<i>Unique?</i>	Yes	No	No	No
<i>Single or Multiple Value</i>	Single	Single	Single	Single
<i>Simple or Composite</i>	Simple	Simple	Simple	Simple

- Candidate keys: docPID
- Primary key: docPID
- Strong/Weak Entity: Strong
- Fields to be indexed: docPID

Subject

<i>subPID</i>	<i>subInitials</i>	<i>subGender</i>	<i>subHeight</i>	<i>subWeight</i>	<i>subBloodType</i>	<i>subStatus</i>	<i>subSurvivalStatus</i>
This ID is used to uniquely identify a Subject.	This contains the first and last letter abbreviations of a Subject's name. HIPAA compliance.	The gender the Subject identifies as is stored here.	The Subject's height, in inches, is stored here.	The Subject's weight, in pounds, is stored here.	The Subject's blood type is stored here.	A Subject is either active, and can participate in Studies, or inactive, and can not.	If a Subject becomes deceased during the course of a Study, that is an important metric.
integer	varchar(75)	varchar(50)	integer	integer	varchar(75)	varchar(50)	integer
All positive integers	String of ASCII maximum 75	String of ASCII maximum 50	All 9-digit positive integers	All positive integers	String of ASCII maximum 75	String of ASCII maximum 50	1 or 0
NASK	None	NULL	NULL	NULL	NULL	1	1
No	No	Yes	Yes	Yes	Yes	No	No
Yes	No	No	No	No	No	No	No
Single	Single	Single	Single	Single	Single	Single	Single
Simple	Simple	Simple	Simple	Simple	Simple	Simple	Simple

- Candidate keys: subPID
- Primary key: subPID
- Strong/Weak Entity: Strong
- Fields to be indexed: subPID

Study

	<i>stuPID</i>	<i>stuName</i>	<i>stuDescription</i>	<i>stuEnrollmentGoal</i>
<i>Description</i>	This ID is used to uniquely identify a Study.	Every Study is assigned a Name, typically referring to the type of drug or technology used.	A Study name may be insufficient for the average person to understand the point of the study, so it is explained in more detail here.	All Studies have a Subject Total goal that they try to reach, that is stored here.
<i>Domain/Type</i>	integer	varchar(50)	varchar(MAX)	integer
<i>Value Range</i>	All positive integers	String of ASCII maximum 50	String of ASCII maximum (SQL MAX)	All positive integers
<i>Default Value</i>	NASK	None	NULL	NULL
<i>Nullable?</i>	No	No	Yes	Yes
<i>Unique?</i>	Yes	No	No	No
<i>Single or Multiple Value</i>	Single	Single	Single	Single
<i>Simple or Composite</i>	Simple	Simple	Simple	Simple

- Candidate keys: stuPID
- Primary key: stuPID
- Strong/Weak Entity: Strong
- Fields to be indexed: stuPID

Study Arm

	<i>sarPID</i>	<i>sarName</i>	<i>sarDescription</i>
<i>Description</i>	This ID is used to uniquely identify a particular Arm of a Study.	This is a label for a particular Arm attached to a Study. (Will normally be "1" or "main")	This field stores a brief description of a Study Arm, it could contain almost anything.
<i>Domain/Type</i>	integer	varchar(50)	varchar(200)
<i>Value Range</i>	All positive integers	String of ASCII maximum 50	String of ASCII maximum 200
<i>Default Value</i>	NASK	None	NULL
<i>Nullable?</i>	No	No	Yes
<i>Unique?</i>	Yes	Yes	No
<i>Single or Multiple Value</i>	Single	Single	Single
<i>Simple or Composite</i>	Simple	Simple	Simple

- Candidate keys: sarPID
- Primary key: sarPID
- Strong/Weak Entity: Strong
- Fields to be indexed: sarPID

Visit

	<i>visPID</i>	<i>visName</i>	<i>visDescription</i>	<i>visStart</i>	<i>visEnd</i>
<i>Description</i>	This ID is used to uniquely identify a Visit	Each Visit will have a (non-unique) label associated with it. Most of the time it will be a number, but not always.	Purely an optional field, populated before or after a Visit if a special note is required (i.e. Subject dies)	This DateTime field holds the estimated start day and time of a particular Visit for a particular set of Subjects.	The estimated ending day and time of a visit is stored here.
<i>Domain/Type</i>	integer	varchar(50)	varchar(100)	DateTime	DateTime
<i>Value Range</i>	All positive integers	String of ASCII maximum 50	String of ASCII maximum 100	Any Date & Time after 1/1/1900	Any Date & Time after 1/1/1900
<i>Default Value</i>	NASK	None	NULL	DateTime(Now)	DateTime(Now)
<i>Nullable?</i>	No	No	Yes	No	No
<i>Unique?</i>	Yes	No	No	No	No
<i>Single or Multiple Value</i>	Single	Single	Single	Single	Single
<i>Simple or Composite</i>	Simple	Simple	Simple	Simple	Simple

- Candidate keys: visPID
- Primary key: visPID
- Strong/Weak Entity: Strong
- Fields to be indexed: visPID

Activity

	<i>actPID</i>	<i>actName</i>	<i>actDescription</i>	<i>actDuration</i>	<i>actCost</i>
<i>Description</i>	This ID is used to uniquely identify an Activity	All Activities are assigned a unique name in Medicine (EKG, Blood Draw, Physical)	If the actName is insufficient to describe an Activity, it may be explained in more detail here	The amount of time (in minutes) that an Activity is expected to take is stored here.	Activities are assigned a cost to be later reported on by Study, by Patient, or by Activity
<i>Domain/Type</i>	integer	varchar(50)	varchar(75)	integer	Decimal(8,4)
<i>Value Range</i>	All positive integers	String of ASCII maximum 50	String of ASCII maximum 75	All positive integers less than 720	Any decimal number 8 digits before the decimal and 4 digits after.
<i>Default Value</i>	NASK	NULL	None	Null	1
<i>Nullable?</i>	No	Yes	No	Yes	No
<i>Unique?</i>	Yes	No	No	No	No
<i>Single or Multiple Value</i>	Single	Single	Single	Single	Single
<i>Simple or Composite</i>	Simple	Simple	Simple	Simple	Simple

- Candidate keys: actPID
- Primary key: actPID
- Strong/Weak Entity: Strong
- Fields to be indexed: actPID

Appointment

	<i>appPID</i>	<i>appStartTime</i>	<i>appEndTime</i>	<i>appCompleted</i>
<i>Description</i>	This ID is used to uniquely identify an Appointment	All Appointments are given a name generated from a combination of values from the Visit & Activity & Subject data it applies to. Inherited from Visit, then modified by User.	All Documents that are selected will have their base filenames saved here. Inherited from Visit, then modified by User.	This is a status field, which should be marked completed at the conclusion of an appointment. When all appointments for a study are completed, the Study is completed
<i>Domain/Type</i>	integer	varchar(50)	DateTime	DateTime
<i>Value Range</i>	All positive integers	String of ASCII maximum 50	Any Date & Time after 1/1/1900	Any Date & Time after 1/1/1900
<i>Default Value</i>	NASK	Inherited	DateTime(Now)	DateTime(Now)
<i>Nullable?</i>	No	No	No	No
<i>Unique?</i>	Yes	Yes	No	No
<i>Single or Multiple Value</i>	Single	Single	Single	Single
<i>Simple or Composite</i>	Simple	Simple	Simple	Simple

- Candidate keys: appPID
- Primary key: appPID
- Strong/Weak Entity: Strong
- Fields to be indexed: appPID

Document Info

	<i>docPID</i>	<i>docDescription</i>	<i>docFileName</i>	<i>docBLOB</i>	<i>docVersion</i>
<i>Description</i>	This ID is used to uniquely identify a Document	This field allows Users uploading Documents to assign a new name to a particular file. The filename is retained in a separate field.	All Documents that are selected will have their base filenames saved here.	This field holds the 9 digit primary phone number of a Sponsor.	This field stores a brief description of the sponsor (to be used if needed)
<i>Domain/Type</i>	integer	varchar(50)	varchar(MAX)	varbinary(MAX)	varchar(200)
<i>Value Range</i>	All positive integers	String of ASCII maximum 50	String of ASCII maximum (SQL MAX)	String of ASCII maximum (SQL MAX)	Any positive integer
<i>Default Value</i>	NASK	NULL	None	Null	1
<i>Nullable?</i>	No	Yes	No	Yes	No
<i>Unique?</i>	Yes	No	No	No	No
<i>Single or Multiple Value</i>	Single	Single	Single	Single	Single
<i>Simple or Composite</i>	Simple	Simple	Simple	Simple	Simple

- Candidate keys: dinPID
- Primary key: dinPID
- Strong/Weak Entity: Strong
- Fields to be indexed: dinPID

1.2.2 Relationship Set Description

Below is a detailed description of all the relationships presented in our ER diagram, as well as some key information about the entities involved and the cardinality of each relationship.

Creates

The Creates relationship connects a User to a Study. Users will create and configure a Study to involve one or many Subjects, Doctors, Activities, Visits, and Appointments.

Mapping cardinality: 1:M

Descriptive field: None

Participation Constraint: Mandatory for Study

Hosts

The Hosts relationship connects a Site to a Study. Most Appointments for a Subject are completed at a Site, rarely it will be held off-site. In that event, a NULL will be recorded.

Mapping cardinality: 1:M

Descriptive field: None

Participation Constraint: Optional for Study

Pays For

The Pays For relationship connects a Sponsor to a Study. All Studies, without exception, are paid for by a single Sponsor. A Sponsor does not necessarily need to have any Studies associated with them, but all Studies require one.

Mapping cardinality: 1:M

Descriptive field: None

Participation Constraint: Mandatory for Study

Has

The Has relationship connects a Study to Document. All Studies, without exception, will have many Documents associated with them. A Study is not required to have a certain number of documents associated with it, but a Document must always be associated with a Study.

Mapping cardinality: 1:M

Descriptive field: None

Participation Constraint: Mandatory for Document

Conducts Studies At

The Conducts Studies At relationship connects a Doctor to a Site. A Doctor may work at various sites, but only participate in a Study at a single Site. A Site does not need to have Doctors associated with it, but a Doctor must be linked to one or more Sites.

Mapping cardinality: 1:M

Descriptive field: None

Participation Constraint: Mandatory for Doctor

Assigned To

The Assigned To relationship connects Doctors and Subjects to a Study. Both Doctors and Subjects may be linked to one or more Studies, but Studies may not link to any Subject or Doctors at the beginning.

Mapping cardinality: M:M

Descriptive field: None

Participation Constraint: Mandatory for Study, Mandatory for Subject, Doctor Mandatory for

Part Of

The Part Of relationship connects Activities and Visits to a Study Arm. Any number of Visits and Activities may be connected to a single Arm, and an Arm can be connected to multiple Visits and Activities.

Mapping cardinality: M:M

Descriptive field: None

Participation Constraint: Mandatory for Visit, Mandatory for Activity, Mandatory for Study Arm

Linked To

The Linked To relationship connects Appointments to a Study Arm. A Study arm may have many appointments, but an Appointment can only be connected to a single Arm.

Mapping cardinality: M:1

Descriptive field: None

Participation Constraint: Mandatory for Appointment

Generated From

The Generated From relationship connects Appointments to Visits and Activities. All Appointments are generated from some Activity scheduled for a certain numbered Visit for a certain Subject. All Appointments must be linked to a Visit & Activity, though if it isn't scheduled, Visits and Activities won't necessarily be used to generate an Appointment. An Appointment can also be added regardless of Visits and Activities

Mapping cardinality: M:1

Descriptive field: None

Participation Constraint: Optional for Appointment

Conducted By

The Conducted By relationship connects Appointments to Doctors. All Appointments are assigned a Doctor who will ensure the Subject completes the Activity. All Appointments have a single Doctor, but a Doctor might have several Appointments.

Mapping cardinality: M:1

Descriptive field: None

Participation Constraint: Mandatory for Appointment

Completed By

The Completed By relationship connects Appointments to Subjects. All Appointments are created for a Subject. A Subject may have multiple Appointments, but an Appointment will only have a single Subject.

Mapping cardinality: M:1

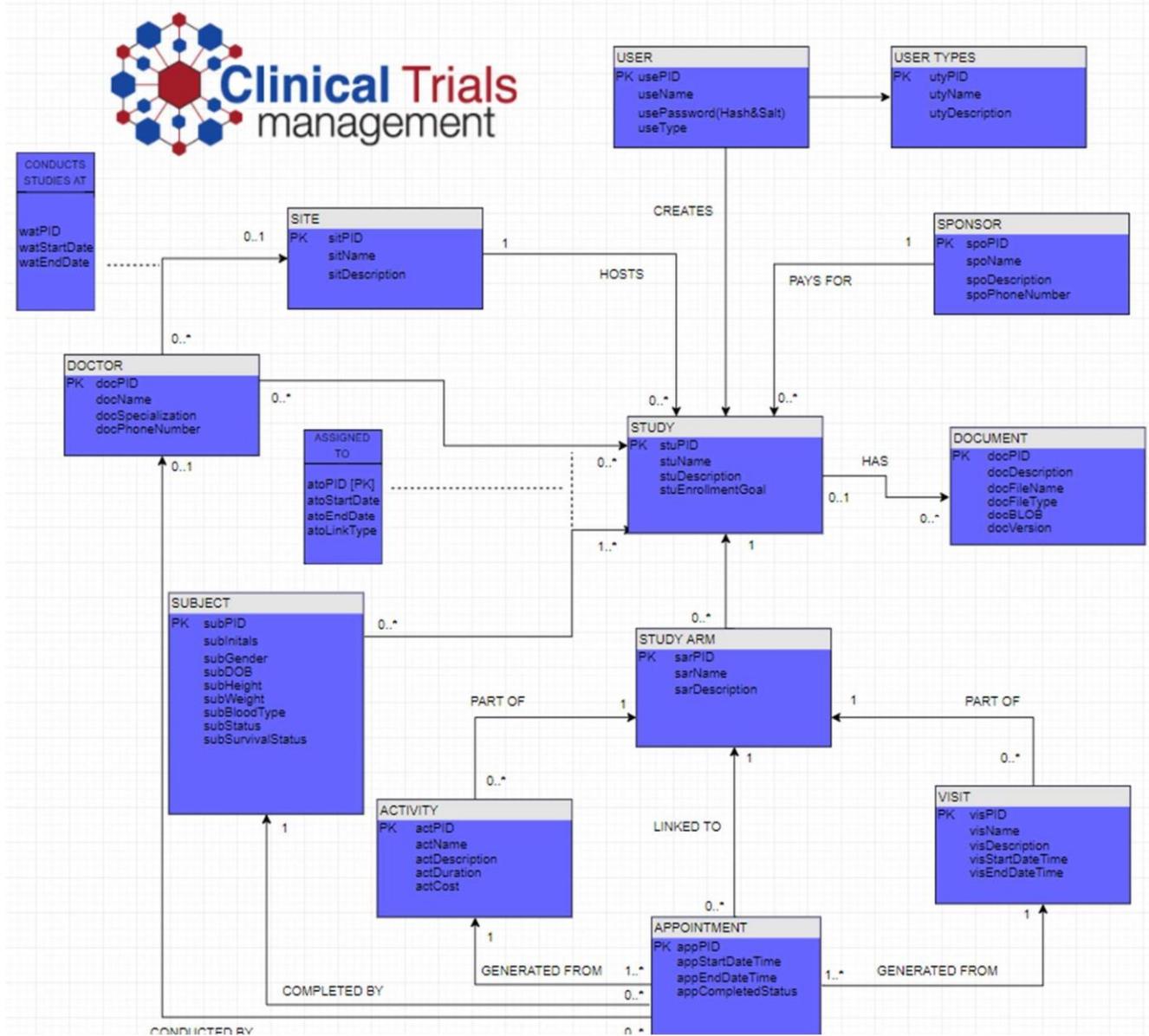
Descriptive field: None

Participation Constraint: Mandatory for Appointment

1.2.3 Related Entity Set

Super and Sub-Classes are not used in our design, instead we've elected to create several related tables that will allow us to manage multiple Studies from any Site, using any Doctors, with any Subjects, paid for by any Sponsor, which could any number of Documents.

1.2.4 ER Diagram



2. Conceptual Database and Logical Database

First we began with the conceptual database which is exactly what the name suggest, it made to give an idea of what the database will look like. The logical database is a translation of the conceptual model but with more implementation in mind. Just like how when a car company unveils a concept, the final product that is available to the public does not look exactly like the concept.

To achieve the logical database, we must convert our E-R model to a relational model. The relational model will be closer to the database itself then the E-R model. Things like

relationships do not exist and their function is replaced by foreign keys and such. After this model is created we will be able to create sample queries that will be used as a foundation for what our database will be able to handle.

2.1 E-R Model and Relational Model

The E-R model describes a conceptual database design. A relational model describes a logical design that will be closer to application, or in other words more realistic. The following will be regarding these two different types of models.

2.1.1 Description of E-R Model and Relational Model

The E-R model was first introduced by a developer named Peter Chen in 1976. Chen was a graduate from National Taiwan University. The main and most important objective of this model is for it to very easily readable and user friendly to everyone. People must be able to look at the model and understand what the idea of this database is. The model is also meant to tie together what the organization wants and what the engineers behind the database wants to do. It is a bridge between the creator of the database and the business it is for.

The relational model was created by Edgar F. Codd in 1969. This model is less about simplicity and more about mirroring the actual way the database will look. It is meant more for the developers of the database. This model is more related to the creation of queries as this model does not have features that are not supported. Everything that is used to create this model can be applied to the vast majority of database management systems. It is structured around the idea of building blocks and those blocks are the relations.

When it comes to comparisons between them there are some that are important to point out. For one, the relational model, as previously stated, relies on the idea of the relation. The E-R model relies on two ideas, the entity, and the relationship. The relationships are formed between the entities to relate them and connect them. Within the entities are attributes which are used to describe the purpose the entity. The relational model relies on the relation and within the relation there may be tuples which could be correlated to the attributes of the E-R model.

2.1.2 Comparison of Two Different Models

There are advantages to the E-R model. The primary one is the ease of understanding. It is basically the purpose the model. It is the ease of communication between developer and customer. The mathematical basis of the relational model is not present in the E-R model. But this is also a disadvantage. Because it does not have a mathematical basis, it can not be used with any formal query language. No database management system(DBMS) does NOT support the ER model. Another issue here is that there is no standard behind the E-R model. This can lead to major issues between developers.

The same can be said about the relational model. One of advantages is one the disadvantages of the E-R model. The relational model does have a standard and it is based on

mathematics. Because of this query languages do support this model. It is far easier to implement the idea of the model into software. But there are disadvantages which are basically the advantages of the E-R model. It is hard to understand unless the person looking it over has prior knowledge in the field. It is not meant to be used to as communication with the customer.

The main difference between the two, however, is the purpose for which they were created for. The E-R model is primarily geared to be user friendly and easier to understand. Due to this fact it is not practical to implement the design behind this model to the software. This is where the relational model comes in. The relation model is meant more for the creator rather than the customer who wants the database. The relational model is closer to the development of the software. Despite these differences, there are similarities. They are both intended to demonstrate the structure that the data must be in. A relation schema is the equal of the entity and relationship types.

2.2 Conversion of Conceptual Database Model to Logical Database Model

Having thoroughly described the both the E-R and the relational model, now it is time to do the conversion from E-R to relational. The first item that will be done is to go through the process of converting the entity types then the relation types. Finally, will go through the process of constructing the constraints that we will need to handle in order to maintain integrity of the database.

2.2.1 Converting Entity Types to Relations

We now have to convert the entity types into relation schemas. This is required when doing a conversion from E-R Model to Relational model. In a sense, relation schemas are much simpler than the complex E-R model because the relation schemas will have single valued domains. This following section will be dedicated to how we will go about this conversion.

When it comes to the conversion of a strong entity into a relation is quite simple. The new relation will retain its name and the attributes will be single value. They will also contain the entity type's composite attributes. One of the key attributes of the entity type will eventually become the primary key of the relation. The remaining key attributes of the entity will simply become candidates. These candidates are unique, but they are not used to identify the relation.

There are also weak entities that must be converted to relations. The way this is done is by making this weak entity into a strong one. The weak entity will contain partial key attributes amongst others. These can be used to form a foreign key that can be used to uniquely identify the relation. If it is still weak then uniqueness must be added to the relation to make it stronger. Once a suitable key is found than the steps above can be followed because this entity is now considered strong.

We must also appropriately handle simple and composite attributes. Simple attributes simply become attributes of the relation. However, composite attributes must be taken apart and only the simple parts of each are used as attributes in the relation. Along with this we must also

handle the attributes that may have single or multi values. As before with simple attributes, single value attributes simply become attributes that will go with the corresponding relation. Multi-value attributes are different. Each multi-value attribute must now become a new relation. Along with this the attributes that go in this new relation are the same ones that were originally in the multi-value attribute. A foreign key is also taken from a relational schema that happens to be a primary key for it. The foreign key and the multi-value attribute are combined to create the primary key the new relation created from the original multi-value variable.

2.2.2 Converting Relationship Types to Relations

Another step in the conversion is the conversion of relationship type to relations. This especially difficult because this concept does not actually exist in the relation model. There are a couple of scenarios that can be found when doing this conversion and here we will cover them.

1:1 Cardinality: When a relationship has a cardinality of 1:1 than both relationship types (A and B) must be converted into relations R1 and R2. Then each instance of R1 must be paired with its match in R2. This can be done in three ways:

1. Merged Relation Approach: This simply brings in both R1 and R2 into one relation.
2. Foreign Key Approach: Here the primary key of R1 is made into a foreign key of R2. Or the other way around. This will ensure that the attributes of the relationship will become attributes of the relation.
3. Cross-Reference Approach: In this approach, a new relation is created. This new relation will contain the primary keys of both R1 and R2 as foreign keys. It will also take the simple attributes of the of the relationship.

1:N Cardinality: As before, the relationship types must be converted into relations, R1 and R2. Each instance of R1 must be related to multiple instances of R2. However, each instance of R2 must be related to one instance of R1. There are two methods to deal with this:

1. Foreign Key Approach: Same as the 1:1 cardinality, however, the foreign key and the attributes must all come from R2. This is so because R2 can only be related to one instance of R1.
2. Cross-Reference Approach: Once again this is the same as the 1:1 cardinality, with the difference that the primary key of the new relation must be the foreign key of the R2 relation.

M:N Cardinality: Once again, we must convert to R1 and R2. However, each instance of R1 can be related to multiple instances of R2. And each instance of R2 can be related to multiple instances of R1. There is only one way to deal with this and it is the Cross-Reference Approach. The primary keys of R1 and R2 are used as foreign keys in the new relation.

“isA” Relationship: This occurs when entity types are disjoint subclasses of one superclass. One entity belongs to on subclass. The three methods to deal with this are as follows:

1. Multiple Relations w/ Superclass & Subclass : Here two relations are needed. One called R_Super and the other called R_Sub. Each will contain the corresponding attributes from the entity. In addition to the attributes found within the R_Sub relation, it will also contain the primary key of the R_Super relation as a foreign key.
2. Multiple Relations w/ Subclass: With this approach, subclass entities are given their own relation. These relations contain a union of the attributes of the superclass and the subclass. If overlapping happens then the appropriate subclasses must all contain the same primary key. (BEST APPROACH)
3. Single Relation w/ One attribute: Only one relation is created and the attributes come from a union of attributes from the one superclass and all the subclasses.

“hasA” Relationship: This occurs when entity types are overlapping subclasses of one superclass. One entity can belong to multiple subclasses. The two methods to mend this are as follows:

1. Multiple Relations w/ Superclass & Subclass: Same exact approach as “isA” relationship
2. Multiple Relations w/ Subclass: With this approach, subclass entities are given their own relation. These relations contain a union of the attributes of the superclass and the subclass. This also contain a Boolean attribute for the subclasses that will indicate “true” if then the instance belongs to that subclass.

When a relationship type associates itself with another relationship type, the best thing that can be done is to create a primary key attribute for the relationship type. Then once both are linked, either the foreign key or the cross-reference approach may be used.

Recursive Relationship: This occurs when one entity type is related to itself. There are two ways deal with this:

1. Foreign Key Approach: The relation will simply have a foreign key that will be the primary key o the relation itself.
2. Cross-Reference Approach: This approach requires the creation of a new relation. This relation will contain two foreign keys that will reference the primary key of the original relation. The combo of both foreign keys will create a primary for the new relation.

When two or more entity types are involved in a relationship, there is a way to deal with it. When this happens, the relationship is turned into one relation. This relation, will contain foreign keys of the other entities’ primary keys. These foreign keys will create the primary key of the new relation.

A union type happens when a subclass belongs to multiple superclasses. These relations that correspond to the different superclasses will have different primary keys, so to mend this a surrogate key must be created. It is also possible to share the same surrogate key in the even that multiple superclasses are associated to one subclass.

2.2.3 Database Constraints

Constraints exist because it is critical for the correct and appropriate data to be inputted. It must have some meaning to it. There are a set of constraints that we cannot go past because it will destroy the integrity of the database. This section will go over the constraints that must be enforced.

Domain Constraints: These constraints revolve around a domain. Each attribute has a domain that will specify what can and cannot be inputted. An example is only accepting certain data types. If this constraint is ignored the DB will reject and could instead set the value to NULL.

Entity Constraint: This constraint requires that all tuples that belong to a particular relation, have a primary key that is not NULL. The reason behind this is to make sure that each tuple can be uniquely identified.

The DBMS can make sure these constraints are ignored by rejecting UPDATE and INSERT operations.

Referential Constraint: This requirement requires that if there exist a reference between two relations using a primary key and a foreign key, then the tuple that was reference must exist. If there is an R1 relation with a tuple T1 that is a foreign key that references another relation R2, then T1 must match T2 which is found somewhere in R2.

Check Constraints and Business Rules: All the constraints that were described above were designed to follow the rules of DBMS. They were constraints that present themselves when there is a conversion between E-R and relation. However, some rules emerge based on the application of the DB. These rules are presented by the business that will be using the DB. These rules will be enforced by the designers to make sure that the users' expectations are followed and that the data satisfies what is needed for the application.

2.3 Convert Entity Relationship Model to Relational Model

2.3.1 Relational Schema for Logical Database Relation Schema: Study

study(stuPID, stuName, stuDescription, stuEnrollmentGoal, stuCompleted, stuSponsorLinkID, stuSiteLinkID) Attributes:

*stuPID	Integer, Primary Key
stuName	Varchar(100)
stuDescription	Varchar(250)
stuEnrollmentGoal	Integer
stuCompleted	Integer(1 or 0)
stuSponsorLinkID	Integer, Foreign Key
stuSiteLinkID	Integer, Foreign Key

Candidate Keys: stuPID, stuSponsorLinkID, stuSiteLinkID

Entity Integrity Constraint: “stuPID” must be unique and not equal to NULL

Uniqueness Constraint: “stuPID” must be unique

Business Constraint: A Study always happens at a Site, but a Study may not always be assigned a Sponsor.

Derivation From Entity and Relationship Types: Study, Sponsor, Site, Document, Subject, Doctor

Study is derived from the Study entity type. Foreign Key's are used to represent the relationship to Sponsor and Site. Documents, Subjects, and Doctors are also assigned to a Study using the “stuPID” as a foreign keys in those relations.

Relation Schema: Study Arm

studyArm(sarPID, sarName, sarDescription, sarStudyLinkID) Attributes:

*sarPID	Integer, Primary Key
sarName	Varchar(250)
sarDescription	Varchar(250)

sarStudyLinkID	Integer, Foreign Key
----------------	----------------------

Candidate Keys: sarPID, sarStudyLinkID

Entity Integrity Constraint: “sarPID” must be unique and not equal to NULL

Uniqueness Constraint: “sarPID” must be unique

Business Constraint: Many Study Arms can be part of the same Study

Derivation From Entity and Relationship Types: Study Arm, Study, Activity, Visit, Appointment

Study Arm is derived from the Study Arm entity type. To represent the relationship between a Study Arm and a Study, a foreign key is used. In the Visit, Activity, and Appointment relations, foreign keys are also used to represent their relationship to a Study Arm. All attributes are single valued.

Relation Schema: Activity activity(actPID, actName, actDescription, actDuration, actCost, actStudyArmLinkID) Attributes:

*actPID	Integer, Primary Key
actName	Varchar(50)
actDescription	Varchar(255)
actDuration	Integer
actCost	Float(12,4)
actStudyArmLinkID	Integer

Candidate Keys: actPID, actStudyArmLinkID

Entity Integrity Constraint: “actPID” must be unique and not equal to NULL

Uniqueness Constraint: “actPID” must be unique

Business Constraint: Many Activities can be linked to the same Study Arm

Derivation From Entity and Relationship Types: Study Arm, Appointment

Activity is derived from the Activity entity type. To represent the relationship between an Activity and a Study Arm, a foreign key is used. A foreign key is also used in the Appointment relation to show the relationship between itself and an Activity. All attributes are single valued.

Relation Schema: Visit

visit(visPID, visName, visDescription, visStartDate, visEndDate, visStudyArmLinkID)

Attributes:

*visPID	Integer, Primary Key
visName	Varchar(50)
visDescription	Varchar(255)
visStartDate	Datetime
visEndDate	Datetime
visStudyArmLinkID	Integer, Foreign Key

Candidate Keys: visPID, visStudyArmLinkID

Entity Integrity Constraint: “visPID” must be unique and not equal to NULL

Uniqueness Constraint: “visPID” must be unique

Business Constraint: Many Visits can be linked to the same Study Arm

Derivation From Entity and Relationship Types: Study Arm, Appointment

Visit is derived from the Visit entity type. To represent the relationship between an Visit and a Study Arm, a foreign key is used. A foreign key is also used in the Appointment relation to show the relationship between itself and an Visit. All attributes are single valued.

Relation Schema: Appointment

appointment(appPID, appName, appDescription, appStartDate, appEndDate, appCompleted, appVisitLinkID, appActivityLinkID, appStudyArmLinkID, appSubjectLinkID, appDoctorLinkID) Attributes:

*appPID	Integer, Primary Key
appName	Varchar(255)
appDescription	Varchar(255)
appStartDate	Datetime
appEndDate	Datetime
appCompleted	Integer(1 or 0)
appVisitLinkID	Integer, Foreign Key
appActivityLinkID	Integer, Foreign Key
appSubjectLinkID	Integer, Foreign Key
appDoctorLinkID	Integer, Foreign Key

Candidate Keys: appPID, appVisitLinkID, appActivityLinkID, appSubjectLinkID, appDoctorLinkID

Entity Integrity Constraint: “appPID” must be unique and not equal to NULL

Uniqueness Constraint: “appPID” must be unique

Business Constraint: None

Derivation From Entity and Relationship Types: Appointment, Visit, Activity, Subject, Doctor

Appointment is derived from the Appointment entity type. Foreign keys are used to show the relation to Visits, Activities, Subjects, and Doctors of an Appointment. All Attributes are single valued.

Relational Schema: Site

site(sitPID, sitName, sitDescription) Attributes:

*sitPID	Integer, Primary Key
sitName	Varchar(100)
sitDescription	Varchar(255)

Candidate Keys: sitPID

Entity Integrity Constraint: “sitPID” must be unique and not equal to NULL

Uniqueness Constraint: “sitPID” must be unique

Business Constraint: None

Derivation From Entity and Relationship Types: Site, Study, Doctor

Visit is derived from the Visit entity type. All attributes are single valued.

Relation Schema:

Doctor

doctor(docPID, docName, docSpecialization, docPhoneNumber) Attributes:

*docPID	Integer, Primary Key
docName	Varchar(255)
docSpecialization	Varchar(255)
docPhoneNumber	Varchar(10)

Candidate Keys: docPID

Entity Integrity Constraint: “docPID” must be unique and not equal to NULL

Uniqueness Constraint: “docPID” must be unique

Business Constraint: A Doctor is assigned to a single Site, but could be working on any number of Studies

Derivation From Entity and Relationship Types: Doctor, Site, Study

Doctor is derived from the Doctor entity type. All attributes are single valued.

Relational Schema: Subject

subject(subPID, subInitials, subGender, subDOB, subHeight, subWeight, subBloodType, subStatus, subSurvivalStatus)

Attributes:

*subPID	Integer, Primary Key
subInitials	Varchar(10)
subGender	Varchar(25)
subDOB	Datetime
subHeight	Float(12,2)
subWeight	Float(12,2)
subBloodType	Varchar(25)
subStatus	Integer(1 or 0)
subSurvivalStatus	Integer(1 or 0)

Candidate Keys: subPID

Entity Integrity Constraint: “subPID” must be unique and not equal to NULL

Uniqueness Constraint: “subPID” must be unique

Business Constraint: Subjects can participate in multiple Studies and be assigned of any number of Appointments.

Derivation From Entity and Relationship Types: Subject, Study, Appointment

Subject is derived from the Subject entity type. All attributes are single valued.

Relational Schema: Sponsor

sponsor(spoPID, spoName, spoDescription, spoPhoneNumber) Attributes:

*spoPID	Integer, Primary Key
spoName	Varchar(255)
spoDescription	Varchar(255)
spoPhoneNumber	Varchar(10)

Candidate Keys: spoPID

Entity Integrity Constraint: “spoPID” must be unique and not equal to NULL

Uniqueness Constraint: “spoPID PID” must be unique

Business Constraint: Sponsors can be assigned to any number of Studies, and is not dependent on a Site.

Derivation From Entity and Relationship Types: Sponsor, Study

Sponsor is derived from the Sponsor entity type. All attributes are single valued.

Document Info

document_info(dinPID, dinDescription, dinFileName, dinFileType, dinBLOB, dinVersion, dinStudyLinkID) Attributes:

*dinPID	Integer, Primary Key
dinDescription	Varchar(255)
dinFileName	Varchar(255)
dinFileType	Varchar(50)
dinBLOB	Varbinary(max)
dinVersion	Integer

dinStudyLinkID	Integer, Foreign Key
----------------	----------------------

Candidate Keys: dinPID, dinStudyLinkID

Entity Integrity Constraint: “dinPID” must be unique and not equal to NULL

Uniqueness Constraint: “dinPID” must be unique

Business Constraint: Studies can have hundreds of Documents

Derivation From Entity and Relationship Types: Document Info, Study

Document Info is derived from the Document Info entity type. A foreign key is used to show Document Info’s relation to a Study. All attributes are single valued.

2.3.2 Sample Data of Relation

Now we will show sets of records (tuples) that correspond to their relational schemas. Attributes will follow the horizontal, and each row will represent an instance of the relation.

Activity						
actPID	actName	actDescription	actDuration	actCost	actStudyArmLinkID	
1	Blood Draw	None	60	50.00	1	
2	MRI	Scanning	120	500.00	2	
3	Examination	Blood pressure, bmi	240	49.99	1	
4	Stress test	Cardio	90	75.00	3	
5	Physical	None	90	19.99	4	
6	Checkup	None	30	0.00	1	

Appointment									
appPID	appStartTime	appEndTime	appCompleted	appStudyArmLinkID	appActivityLinkID	appVisitLinkID	appSubjectLinkID	appDoctorLinkID	appVisitLinkID
1	2018-04-23 14:10	2018-05-10 0:47:7		1	1	1	1	1	1
2	2018-04-18 7:45:6	2018-04-24 15:52:22		1	1	1	6	2	3
3	2018-04-11 13:10:30	2018-04-02 0:43:43	0	3	3	2	4	5	
4	2018-04-30 9:40:8	2018-04-28 21:6:11	0	4	1	4	3	2	
5	2018-05-02 6:42:27	2018-05-28 11:6:25		1	2	2	3	3	3
6	2018-04-23 6:36:1	2018-05-10 9:16:53		1	1	1	1	1	4
7	2018-05-07 23:64:2	2018-04-16 23:36:48	0	2	3	4	2	2	
8	2018-04-21 19:24:25	2018-04-08 7:22:51		1	2	3	6	2	1
9	2018-04-28 17:40:26	2018-04-21 12:18:14		1	3	1	5	5	5
10	2018-04-24 21:26:37	2018-05-26 9:48:58		1	4	4	1	1	2

Assigned To							
atoPID	atoStartDate	atoEndDate	atoLinkType	atoSubjectLinkID	atoDoctorLinkID	atoStudyLinkID	atoVisitLinkID
1	2018-05-20 10:20:25	2018-05-07 2:25:35		1	1	NULL	1
2	2018-04-05 10:9:51	2018-04-22 5:10:37		1	2	NULL	4
3	2018-05-24 4:41:28	2018-05-19 3:28:22		1	3	NULL	3
4	2018-05-30 5:28:52	2018-04-26 15:26:44		2	NULL	1	1
5	2018-04-17 18:25:20	2018-05-23 22:11:53		2	NULL	4	4
6	2018-05-14 20:47:13	2018-04-21 5:13:2		2	NULL	2	2
7	2018-05-07 3:16:55	2018-05-19 3:54:54		1	1	NULL	3
8	2018-04-16 11:33:6	2018-05-17 18:44:40		1	4	NULL	3
9	2018-05-01 19:53:55	2018-04-08 17:26:35		2	NULL	3	4
10	2018-04-26 13:27:47	2018-05-22 0:15:27		2	NULL	3	2

Conducts Studies At				
cdaPID	cdaStartDate	cdaEndDate	cdaDoctorLinkID	cdaSiteLinkID
1	2018-05-20 14:19:33	2018-04-03 23:29:35	1	5
2	2018-05-01 12:53:4	2018-04-18 12:32:50	2	4
3	2018-05-17 4:15:48	2018-04-24 6:31:31	4	3
4	2018-05-10 21:11:30	2018-04-25 6:54:12	3	1
5	2018-04-13 20:27:44	2018-04-08 12:30:43	5	2

Doctor				
docPID	docName	docSpecialization	docPhoneeNumber	
1	Tweaty	Speech Patterns	1236781237	
2	Bugs	Olfactory Nerves	4803475879	
3	Daffy	Study of Ornithoids	3579810025	
4	Elmer	None	4784268987	
5	Roadrunner	Cardiovascular Health	4602167851	

Document						
docPID	docDescription	docFileName	docFileType	docBLOB	docVersion	docStudyLinkID
1	Receipts	receipts.pdf	"PDP"	3423DSAi+...	1	1
2	Regulatory Document	regulatoryDocs.doc	"Doc"	33jsDSAi+...	1	3
3	Receipts	receipts.pdf	"PDP"	54d3DSAi+...	2	1
4	Visit Log	visitlog.pdf	"PDP"	pA+dDSAi+...	1	2
5	Visit Log	visitlog.pdf	"PDP"	3423DSAi+...	2	2
6	Receipts	receipts.pdf	"PDP"	3423DSAi+...	3	1
7	Receipts	receipts.pdf	"PDP"	pE3+DSAi+...	4	1
8	Regulatory Document	regulatoryDocs.doc	"Doc"	3423DSAi+...	1	2
9	Regulatory Document	regulatoryDocs.doc	"Doc"	3423DSAi+...	1	4
10	Receipts	receipts.pdf	"PDP"	43sDL+Ai+...	5	1

Site		
sitPID	sitName	sitDescription
1	Redlands Hospital	Hospital
2	Bob's Clinic	Clinic
3	Beaumont Hospital	Hospital
4	Bakersfield ER	Emergency services
5	LA Hospital	Hospital

Sponsor

spoID	spoName	spoDescription
1	Really Big Pharma	None
2	Much Larger Pharma	None
3	Biggest Pharma	Cash Cow

Study

stuID	stuName	stuDescription	stuEnrollmentGoal
1	New Drug	Neosporin 2.0	20
2	Old Drug	Advil	15
3	New Technology	Heart rate monitor	75
4	Old Technology	Diabetes test kit	25

Study Arm

sarID	sarName	sarDescription	sarStudyLinkId
1		Main arm	1
2		Secondary arm	1
3		Main arm	2
4		Secondary arm	2

Subject

subID	subName	subGender	subDOB	subHeight	subWeight	subBloodType	subStatus	subSurvivalStatus
1	Mike	Male	2010-05-18 79:52	72	135	A+	1	1
2	Jessica	Female	2010-04-29 12:43:13	78	194	B	1	1
3	James	Male	2005-04-30 19:55:54	67	210	O	1	1
4	Bobby	Male	2010-04-22 14:33:39	54	188	B-	1	1
5	Felicity	Female	2010-04-27 23:43:1	87	140	A-	1	1
1	Semore	Not Disclosed	2001-04-06 36:13	64	167	A-	1	1
2	Weird Al	Not Disclosed	2010-05-03 21:32:25	68	152	B+	1	1
3	Hailey	Female	2010-05-09 7:11:38	70	174	O	0	1
4	Halley	Female	2010-04-20 17:0:2	62	185	O	0	1
5	Elvis	Male	1930-05-02 23:27	58	138	B+	1	0

Visit	visPID	visName	visDescription	visStartTime	visEndTime	visStudyArmLinkID
	1	1	None	2018-04-24 3:29:21	2018-05-01 3:5:51	1
	2	2	Rescheduled	2018-04-13 6:55:16	2018-05-15 18:50:46	2
	3	1	None	2018-05-08 5:55:51	NULL	1
	4	2	None	2018-04-20 15:47:16	NULL	3
	5	3	None	2018-04-07 3:22:13	2018-04-02 6:52:9	4
	6	1	None	2018-04-11 19:58:5	NULL	1

2.4 Sample Queries

These are some of the queries we came up with:

- 1) How many appointments were completed between 1/1/2018 and 2/1/2018 for currently completed studies?
- 2) How many sites have more than two doctors?
- 3) How many subjects have an appointment scheduled for 4/20/2018?
- 4) How many doctors are participating in two currently incomplete studies?
- 5) What is the least expensive completed/incomplete study?
- 6) What studies have at least two assigned activities?
- 7) Which doctors have never had an appointment with “John Doe”?
- 8) Which doctors have seen at least two patients at an appointment?
- 9) List sponsors involved in at least 2 completed studies.
- 10) Which subjects have been involved in a study that has been completed?

2.4.1 Design of Queries

In this section we will answer the queries specified above in Relational Algebra, Tuple Relational Calculus, and Domain Relational Calculus. Each of the languages will be explained beforehand.

2.4.2 Relational Algebra Expressions for Queries

Relational algebra is used to retrieve rows or records from a relational database. Relational algebra is a procedural language, and multiple expressions are combined to answer important and/or complex queries.

- 1) How many appointments were completed between 1/1/2018 and 2/1/2018 for currently completed studies?

A handwritten relational algebra expression on a piece of paper. The expression is:

$$\Pi_{a.appPID} \left(\sigma_{\begin{array}{l} a.appointment \bowtie s \\ \begin{array}{l} a.appStartDate \geq 1/1/2018 \wedge \\ a.appEndDate \leq 2/1/2018 \wedge \\ a.appCompleted = 1 \wedge \\ s.StuCompleted = 1 \end{array} \end{array}} \right)$$

- 2) How many sites have more than two doctors?

$$\Pi_{S.SitName} \left(\sigma_{\substack{D1 \\ doctor * doctor}} \bowtie^S Site \right)$$

$$\left(D1.docSiteLinkID = D2.docSiteLinkID \wedge D1.docPID \neq D2.docPID \right)$$

- 3) How many subjects have an appointment scheduled for 4/20/2018?

$$\Pi_{S.SubPID} \left(\sigma_{\substack{S \\ Subject * appointment}} \right)$$

$$S.SubPID = a.appSubjectLinkID \wedge a.appStartDate = 4/20/2018$$

- 4) How many doctors are participating in two currently incomplete studies?

$$\Pi_{d.docName} \left(\sigma_{\substack{S1 \\ Study * Study}} \bowtie^a \text{appointment} \bowtie^d \text{doctor} \right)$$

$$\left(S1.StuPID \neq S2.StuPID \wedge S1.StuCompleted = 0 \wedge S2.StuCompleted = 0 \wedge a.appDoctorLinkID = d.docPID \right)$$

- 5) What is the least expensive completed/incomplete study?

$$\nexists_{a.actPID} \left(\sigma_{a.actPID}(\text{activity}) - \sigma_{a.actPID}^{a_1, a_2}(\text{activity} * \text{activity}) \right) \\ (\text{a}_1.\text{act}(\text{lost} \rightarrow \text{a}_2.\text{act}(\text{lost} \wedge \text{a}_1.\text{actPID} \neq \text{a}_2.\text{actPID}))$$

- 6) What studies have at least two assigned activities?

$$\nexists_{S_1.\text{stuPID}} \left(\sigma_{S_1.\text{stuPID}} \left(\sigma_{S_1.\text{stuName}} \left(\sigma_{S_1.\text{stuDescription}} \left(\sigma_{\text{Activity} * \text{Activity}} \times \text{StudyArm} * \text{Study} \right) \right) \right) \right) \\ A_1.\text{actPID} \neq A_2.\text{actPID} \wedge \\ A_1.\text{actStudyArmLinkID} = A_2.\text{actStudyArmLinkID} \wedge \\ A_1.\text{actStudyArmLinkID} = S_1.\text{stuPID} \wedge \\ S_1.\text{stuStudyLinkID} = S_1.\text{stuPID}$$

- 7) Which doctors have never had an appointment with "John Doe"?

$$T_1 \leftarrow \sigma_{A.\text{appDoctorLinkID} = D.\text{docPID} \wedge A.\text{appSubjectLinkID} = S.\text{subPID} \wedge S.\text{subName} = "John Doe"} \left(((\text{Appointment} * \text{Doctor}) * \text{Subject}) \right)$$

$$T_2 \leftarrow \left(\sigma_{A.\text{appDoctorLinkID} = D.\text{docPID} \wedge A.\text{appSubjectLinkID} = S.\text{subPID}} \left(((\text{Appointment} * \text{Doctor}) * \text{Subject}) \right) \right) - T_1$$

Result $\left(\begin{array}{c} \text{docPID} \\ \text{docName} \end{array} \right) (T_2)$

- 8) Which doctors have seen at least two patients at an appointment?

$\Pi_{D.docP1D, D.docName} ($ $\sigma_{(Appointment \bowtie Appointment) \times Doctor}$
 $A_1.appDoctorLinkID = A_2.appDoctorLinkID \wedge$
 $A_1.appSubjectLinkID \neq A_2.appSubjectLinkID \wedge$
 $A_1.appDoctorLinkID = D.docP1D$

- 9) List sponsors involved in at least 2 completed studies.

$\Pi_{S.spoP1D, S.spoName} ($ $\sigma_{((Stu \bowtie Study) \times Sponsor) \wedge$
 $Stu_1.stuP1D \neq Stu_2.stuP1D \wedge$
 $Stu_1.stuCompleted = 1 \wedge$
 $Stu_1.stuCompleted = Stu_2.stuCompleted \wedge$
 $Stu_1.stuSponsorLinkID = Stu_2.stuSponsorLinkID \wedge$
 $Stu_1.stuSponsorLinkID = S.spoP1D}$

- 10) Which subjects have been involved in a study that has been completed?

$\Pi_{Su.subP1D, Su.subjectName} ($ $\sigma_{(Subject \times AssignedTo \times Study) \wedge$
 $A.atolinkType = "Subject" \wedge$
 $A.atoSubjectLinkID = Su.subP1D \wedge$
 $A.atoStudyLinkID = Stu.stuP1D \wedge$
 $Stu.stuCompleted = 1}$

2.4.3 Tuple Relational Calculus Expressions for Queries

Relational calculus operations describe the rows or tuples that are retrieved. The expressions are non-procedural, so the order of operations isn't taken into consideration. Free and bound variables are used to extract tuples, which is the data that fits on the specified domain.

- How many appointments were completed between 1/1/2018 and 2/1/2018 for currently completed studies?

$$\{ a_1 \left| \begin{array}{l} \text{appointment}(a_1) \wedge \exists a_2 \left(\text{appointment}(a_2) \right) \wedge \left(\begin{array}{l} a_1.\text{appCompleted} = 1 \wedge \\ a_2.\text{appCompleted} = 1 \wedge a_1.\text{AppStartDate} = 1/1/2018 \wedge \\ a_2.\text{appEndDate} = 1/1/2018 \wedge a_1.\text{appPID} \neq a_2.\text{appPID} \end{array} \right) \end{array} \right. \}$$

- How many sites have more than two doctors?

$$\{ D_1 \left| \begin{array}{l} \left(\text{doctor}(D_1) \wedge \text{doctor}(D_2) \wedge \text{site}(s) \right) \wedge \left(\begin{array}{l} D_1.\text{docSiteLinkID} = D_2.\text{docSiteLinkID} \wedge \\ D_1.\text{docPID} \neq D_2.\text{docPID} \end{array} \right) \end{array} \right. \}$$

- How many subjects have an appointment scheduled for 4/20/2018?

$$\{ s \left| \begin{array}{l} \text{subject}(s) \wedge \left(\exists a \right) \left(\text{appointment} \right) \wedge \left(\begin{array}{l} s.\text{SubPID} = a.\text{appSubjectLinkID} \wedge \\ a.\text{appStartDate} = 4/20/2018 \end{array} \right) \end{array} \right. \}$$

- 4) How many doctors are participating in two currently incomplete studies?

$$\{d \mid \text{doctor}(d) \wedge (\exists s_1)(\exists s_2) \left(\text{study}(s_1) \wedge \text{study}(s_2) \right) \wedge (\exists a) \left(\text{appointment}(a) \right) \wedge \\ \left(s_1.\text{stuPID} \neq s_2.\text{stuPID} \wedge s_1.\text{stuCompleted} = 0 \wedge s_2.\text{stuCompleted} = 0 \wedge \right. \\ \left. a.\text{appDoctorLinkID} = d.\text{docPID} \right)$$

- 5) What is the least expensive completed/incomplete study?

$$\{a_1 \mid \text{activity}(a_1) \wedge \exists a_2 \left(\text{activity}(a_2) \right) \wedge \\ \left(a_1.\text{actCost} \leq a_2.\text{actCost} \wedge \right. \\ \left. a_1.\text{actPID} \neq a_2.\text{actPID} \right)$$

- 6) What studies have at least two assigned activities?

$$\{s \mid \text{study}(s) \wedge (\exists sa) \left(\text{studyArm}(sa) \right) \wedge \exists a_1 \left(\text{Activity}(a_1) \right) \wedge \exists a_2 \left(\text{Activity}(a_2) \right) \wedge \\ \wedge \left(a_1.\text{actPID} \neq a_2.\text{actPID} \wedge a_1.\text{actStudyArmLinkID} = a_2.\text{actStudyArmLinkID} \wedge \right. \\ \left. a_1.\text{actStudyArmLinkID} = sa.\text{sarPID} \wedge sa.\text{sarStudyLinkID} = s.\text{stuPID} \right)$$

}

- 7) Which doctors have never had an appointment with "John Doe"?

$$\{ D \mid \text{Doctor}(D) \wedge \neg (\exists A_1(\text{Appointment}(A_1) \wedge \exists A_2(\text{Appointment}(A_2) \wedge \exists S(\text{Subject}(S) \wedge A_1.\text{appPID} \neq A_2.\text{appPID} \wedge A_1.\text{appSubjectLinkID} = A_2.\text{appSubjectLinkID} \wedge A_1.\text{appSubjectLinkID} = S.\text{subPID} \wedge S.\text{subName} = "John Doe"))) \}$$

- 8) Which doctors have seen at least two patients at an appointment?

$$\{ D_1 \mid \text{Doctor}(D_1) \wedge \exists A_1 \exists A_2 \exists S_1 (\text{Appointment}(A_1) \wedge \text{Appointment}(A_2) \wedge \text{Subject}(S_1) \wedge A_1.\text{appPID} \neq A_2.\text{appPID} \wedge A_1.\text{appSubjectLinkID} \neq A_2.\text{appSubjectLinkID} \wedge A_1.\text{appDoctorLinkID} = A_2.\text{appDoctorLinkID} \wedge A_1.\text{appDoctorLinkID} = D_1.\text{docPID}) \}$$

- 9) List sponsors involved in at least 2 completed studies.

$$\{ Sp \mid \text{Sponsor}(Sp) \wedge \exists St_1 \exists St_2 (\text{Study}(St_1) \wedge \text{Study}(St_2) \wedge (St_1.\text{stuPID} \neq St_2.\text{stuPID} \wedge St_1.\text{stuSponsorLinkID} = St_2.\text{stuSponsorLinkID} \wedge St_1.\text{stuSponsorLinkID} = Sp.\text{spoPID} \wedge St_1.\text{stuCompleted} = St_2.\text{stuCompleted} \wedge St_1.\text{stuCompleted} = 1)) \}$$

- 10) Which subjects have been involved in a study that has been completed?

$\{ \text{Su} \mid \text{Subject}(\text{Su}) \wedge \exists_{\text{At}} \exists_{\text{St}} (\text{Assigned To}(\text{At}) \wedge \text{Study}(\text{St}) \wedge$
 $(\text{At}. \text{subjectLinkID} = \text{Su}. \text{subP/D}) \wedge (\text{At}. \text{atStudy LinkID} = \text{St}. \text{stuP/D}) \wedge$
 $(\text{St}. \text{stuCompleted} = 1)) \}$

2.4.4 Domain Relational Calculus Expressions for Queries

Domain Relational calculus can best be described as a different kind of relational calculus. Variables represent data or values of a tuple (the individual instances of an attribute) as opposed to the entire tuple.

- How many appointments were completed between 1/1/2018 and 2/1/2018 for currently completed studies?

$$\{ \langle i, s, e, c \rangle \mid \text{appointment}(-, i, -, s, E, C, -, -, -, -, -, -) \wedge \exists a \text{ app}(\exists i_e)$$

$$\text{appointment}(-, i_2, 1 | 2018, 2 | 2018, 1, -, -, -, -, -, -)$$

$$\}$$

- How many sites have more than two doctors?

$$\{ \langle i, n, -, -, s_i \rangle \mid \text{doctor}(i, n, -, -, s_i) \wedge \exists \text{site}(i, n, -) \wedge (\exists d_2)$$

$$\text{doctor}(i \neq d_2, -, -, -)$$

$$\}$$

- How many subjects have an appointment scheduled for 4/20/2018?

$$\{ \langle i, -, -, -, -, -, -, -, - \rangle \mid \text{subject}(i, -, -, -, -, -, -, -) \wedge$$

$$(\exists a) (\text{appointment}(-, a, -, 4 | 20 | 2018, -, -, -, i, -, -))$$

4) How many doctors are participating in two currently incomplete studies?

$$\{ \langle i, n, -, -, -, a_i \rangle \mid \text{doctor}(i, n, -, -, -, a_i) \wedge (\exists a_{iz}) \\ (\text{appointment}(a_{iz} = a_i, -, -, -, -, -, -, -, -) \wedge \\ (\exists s_1)(\exists s_2)(\text{study}(s_1 \neq s_2, -, -, -, -, -, -)) \}$$

5) What is the least expensive completed/incomplete study?

$$\{ \langle i, -, -, -, c, - \rangle \mid \text{activity}(i, -, -, -, c, -) \wedge (\exists i_2)(\exists c_2) \\ \text{activity}(i \neq i_2, -, -, -, c_1 < c_2, -)$$

6) What studies have at least two assigned activities?

$$\{ \langle i, s, e, c \rangle \mid \text{appointment}(-, i, -, s, e, c, -, -, -, -, -, -) \wedge \text{appointment}(-, i_2, z | z | 2018, z | z | 2018, 1, -, -, -, -, -, -) \\ \}$$

7) Which doctors have never had an appointment with “John Doe”?

$$\{ \langle i, n, -, -, s_i \rangle \mid \text{doctor}(i, n, -, -, s_i) \wedge \exists \text{site}(i, n, -) \wedge (\exists d_2) \\ \text{doctor}(i \neq d_2, -, -, -) \\ \}$$

8) Which doctors have seen at least two patients at an appointment?

$$\{ \langle i, - , - , - , - , - , - , - \rangle \mid \text{subject}(i, - , - , - , - , - , -) \wedge \\ (\exists a)(\text{appointment}(-, a, -, 4 | 20 | 2018, - , - , i, - , -)$$

9) List sponsors involved in at least 2 completed studies.

$$\{ \langle i, n, - , - , - , a_i \rangle \mid \text{doctor}(i, n, - , - , - , a_i) \wedge (\exists a_{i2}) \\ (\text{appointment}(a_{i2} = a_i, - , - , - , - , - , - , i, -) \wedge \\ (\exists s_1)(\exists s_2)(\text{study}(s_1 \neq s_2, - , - , - , - , - , -))$$

10) Which subjects have been involved in a study that has been completed?

$$\{ \langle i, - , - , - , c, - \rangle \mid \text{activity}(i, - , - , - , c, -) \wedge (\exists i_2)(\exists c_2) \\ \text{activity}(i \neq i_2, - , - , - , c_1 < c_2, -)$$

3. SQL Server and SQL Server Management Studio

3.1 Normalization of relations

Before we can implement a physical version of our logical database, we must first make sure that the quality of the relational database is adequate. This part of phase three is dedicated to analyzing the quality of our relation database using a method called normalization. This will survey the schema to make sure that it is well designed. This will find potential issues of the design that may occur when applying certain operations to a poorly designed relation database design once the change is made to a logical database.

3.1.1 Normalization and Normal Forms

There are certain levels of normalization and this section will cover them. A relational database schema can be designed in such a way that tuples may contain data that is redundant. This will result in inaccurate result and even errors because if that tuple is changed it may also have to be changed in several others for the integrity of the data to remain secure. This is known as a modification anomaly.

Normalization consists of breaking down relation schemas for the sole purpose to eliminate redundancy and these modification anomalies do not happen.

Normal form tests are designed to test relational schemas to make sure they are normalized. Normal form(NF) tests have four levels: first, second, third, and Boyce-Codd. They are also ranked with first being the least desirable and Boyce-Codd being the most desirable. Typically all schemas should fall within the upper half of the spectrum. Next, each level will be described:

First Normal Form

To achieve First Normal Form(1NF) then all attributes in a relation schema must be single and atomic. They must NOT have tuples that have nested tuples of values. To combat this, multi-value attributes are broken down into separate relations that have original primary key as a foreign key attribute. This method was also described previously in Phase two when we covered how to handle multi-value attributes. Other solution to this involve creating new tuples that will have levels to them, creating tuples that will depend on other tuples, this should be avoided because it will ultimately lead to a modification anomaly.

Second Normal Form

To achieve Second Normal Form(2NF) it must first satisfy 1NF and also, all attributes are have nothing to do with the primary key, must fully functionally depend on that primary key. All other attributes other than the primary key, revolve around that primary key. The concept is simple. A set Y depends on an another set of values X, this is so if and only if X maps to the set of values of Y. In other words, Y depends on X.

This leads to the non-prime attributes to functionally depend on the entire primary key and not just a part of it, not matter how bug the part. If a relation schema has a single attribute primary key, then it automatically passes the 2NF test.

Should a schema fail this test, then it must be broken down into smaller relation schema who have primary keys that are direct subsets of the primary that was originally part of the bigger relation schema.

Third Normal Form

Third Normal Form(3NF) is much simpler than the previous forms. First and foremost, it must satisfy the first two forms and it must also have its non-prime attributes NOT depend on the primary key. In other words, all the attributes in a table are determined only by candidate keys. To solve a failed 3NF test, the schema must be broken down into other relations where the left side of the functional dependency is ALWAYS the primary key attribute.

This completes the first three forms, with one to go. But it is a good place to mention a handy phrase that is spin on another already famous phrase. “... the key, the whole key, and nothing but the key.” “The key”(1NF) ensures that it does in fact exist, “the whole key”(2NF) ensures that non-prime attributes depend on the key, and finally “nothing but the key” (3NF) ensures that it is in fact the most important piece of the puzzle.

Boyce-Codd Normal Form

The Boyce-Codd Normal Form(BCNF) requires that all previous forms be satisfied. It is very similar to 3NF but is far more advanced and has a higher standard. It requires that the left side of any functional dependency must be a primary key. In addition to this it does NOT allow any of the prime attributes to depend on non-prime attributes. To mend this issue, it is possible to break down the relation where non-prime attributes located on the left side become prime attributes of a new relational schema. Anomalies from poor Normalization

The upper half of the forms must be satisfied in order to make sure that redundancy does not happen. There are three anomalies that may happen:

Update Anomalies

If there is a join of a set of tuples of a single relation to several other relations, then the values of the attributes that are part of the single relation will appear in all the tuples after the join. If any of the values are changed in any one tuple than they must be changed for all in order to maintain integrity amongst the data.

Insertion Anomalies

When two relations are joined using natural join and stored into one relation, this creates redundant data. Assuming that the first three forms were compromised. When trying to insert new tuples for the newly relation, anomalies can occur. In order to insert two tuples that both represent two different relations into one single relation, the attribute values must be the same in order for the data to not become corrupted. And when trying to insert a tuple that represents a relation that is NOT joined to any other relations, the attributes of that relation schema must ALL

be set to NULL. And if any of the attributes that help create the primary key of the joined relation are set to NULL, then the integrity constraint is now compromised.

Deletion Anomalies

When a single relation that is represented by a set of tuples is joined to several other relations, and then all of these tuples are removed then any part of the single relation will be removed from the database.

3.1.2 Normal Forms for Our Own Database

USER:

FD = Functional Dependency

FD1. Trivial: {stuPID} -> (stuDescription, stuEnrollmentGoal...,stuSiteLinkID)

FD2.{userPID}

Candidate Keys:

-userPID

Normal Form Checklist:

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). SITE:

FD1. Trivial: {sitPID}-> (sitName,
sitDescription) FD2. Trivial: {sitName}-

>(sitPID, sitDescription) Candidate Keys:

sitPID, sitName

Normal Form Checklist:

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).

SPONSOR:

FD1. Trivial: {spoPID} -> (spoName, spoDescription, spoPhoneNumber)

FD2. Trivial: {spoName} -> (spoName, spoDescription, spoPhoneNumber) Candidate Keys: spoPID, spoName

Normal Form Checklist:

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).

DOCTOR:

FD1. Trivial: {docPID} -> (docName, docSpecialization, docPhoneNumber)

Candidate Keys:

docPID

Normal Form Checklist:

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).

SUBJECT:

FD1. Trivial: {subject} -> (subPID, subInitials, subGender, subDOB, ..., subSurvivalStatus)

Candidate Keys:

subPID

Normal Form Checklist:

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).

STUDY

FD1. Trivial: {stuPID} -> (stuName, stuDescription, stuEnrollmentGoal, ..., stuSiteLinkID)

Candidate Keys:

stuPID

Normal Form Checklist:

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).

STUDY ARM

FD1. Trivial: {sarPID} -> (sarName, sarDescription, sarStudyLinkID)

Candidate Keys:

sarPID

Normal Form Checklist:

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).

VISIT

FD1. Trivial: {visPID} -> (visName, visDescription, visStartDate, visEndDate, visStudyArmLinkID) Candidate Keys:

visPID

Normal Form Checklist:

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).

ACTIVITY

FD1. Trivial: {actPID} -> (actName, actDescription, actDuration, actCost, actStudyArmLinkID)

Candidate Keys:

actPID

Normal Form Checklist:

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).

APPOINTMENT

FD1. Trivial: {appPID} -> (appName, appDescription, ..., appSubjectLinkID, appDoctorLinkID)

Candidate Keys:

appPID

Normal Form Checklist:

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).

DOCUMENT INFO

FD1. Trivial: {dinPID} -> (dinDescription, dinFileName, ..., dinStudyLinkID)

Candidate Keys:

dinPID

Normal Form Checklist:

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).

3.3 Schema Objects for SSMS

Tables:

Tables are the bottom layer of data storage in a SQL Database. Tables are composed of a certain kind of schema which includes the column name and a datatype (Varchar, Nvarchar, int, DateTime). After data is inserted it can remain there unchanged forever or be updated or deleted using SQL Update and Delete statements. Additionally, the data can be retrieved with queries using a Select statement. Virtual (or temporary) tables can also be created based upon the schema of existing tables, or be given a unique schema on the fly.

Syntax:

```
--Simple CREATE TABLE Syntax (common if not using options)
CREATE TABLE
    [ database_name . [ schema_name ] . | schema_name . ] table_name
        ( { <column_definition> } [ ,...n ] )
    [ ; ]
```

Views:

Views are the output of stored queries which do not in themselves store any real data but rather return the most up to date data already stored. Select statements are used

to join together tables to show whatever data is required. This tool allows a developer or database administrator to control what data is shown and how as opposed to giving the average user access to data he/she may not be allowed to see. Views are most commonly returned to front end applications to show stored that stored data.

Syntax:

```
CREATE VIEW "VIEW_NAME" AS "SQL Statement";
```

OR

Create or Replace View <View_Name> As <Select...>

Stored Procedures:

A stored procedure is just a pre-written section of reusable (and likely frequently used) SQL code that performs an action (Select/Update/Delete). Since stored procedures can be called with triggers they can also be automatically called whenever an event occurs on the data. It's good practice to call stored procedures from a front-end of an application to perform operations as opposed to interacting with the data directly.

Syntax:

```
CREATE [ OR ALTER ] { 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 ]
```

Triggers:

A trigger is just a pre-written section of SQL that runs each time data is updated, inserted or deleted (as defined). They can also be created on other kinds of events like when a user logs in.

Syntax:

```

CREATE [ OR ALTER ] TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS { sql_statement [ ; ] [ ,...n ] | EXTERNAL NAME <method specifier [ ; ] > }

<dml_trigger_option> ::=
[ ENCRYPTION ]
[ EXECUTE AS Clause ]

<methodSpecifier> ::=
assembly_name.class_name.method_name

```

Packages:

Packages are collections of objects and stored procedures that encapsulate data and functions for use in front-end applications without giving complete access to the code behind it. There is a “public” interaction with the package as well as the full listing of all functions. Oracle accomplishes this more natively than SQL Server.

Syntax: (Oracle)

Create Package <Package_Name> as <Declarations> end

Sequence Generator:

A sequence object is a schema-bound object that generates a series of values (increasing/decreasing) at predefined intervals. Sequences are not associated with specific tables, but rather a user application can reference a sequence object and use those values across multiple tables. There are also functions (like GET VALUE FOR) that can be used to get sequence information without inserting rows.

Syntax:

```

CREATE SEQUENCE [schema_name . ] sequence_name
[ AS [ built_in_integer_type | user-defined_integer_type ] ]
[ START WITH <constant> ]
[ INCREMENT BY <constant> ]
[ { MINVALUE [ <constant> ] } | { NO MINVALUE } ]
[ { MAXVALUE [ <constant> ] } | { NO MAXVALUE } ]
[ CYCLE | { NO CYCLE } ]
[ { CACHE [ <constant> ] } | { NO CACHE } ]
[ ; ]

```

Index:

An index allows queries to be completed much faster than those with reference tables without them. An index can be based on a unique set of columns or as a single unique column. Indexes can be created/recreated at will and do not affect the data inside the table being referenced.

Syntax:

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX index_name
    ON <object> ( column [ ASC | DESC ] [ ,...n ] )
    [ INCLUDE ( column_name [ ,...n ] ) ]
    [ WHERE <filter_predicate> ]
    [ WITH ( <relational_index_option> [ ,...n ] ) ]
    [ ON { partition_scheme_name ( column_name )
        | filegroup_name
        | default
    }
    ]
    [ FILESTREAM_ON { filestream_filegroup_name | partition_scheme_name | "NULL" } ]
[ ; ]

<object> ::=

{  
    [ database_name. [ schema_name ] . | schema_name. ]
    table_or_view_name
}
```

3.4.1 List Relations with SQL commands

Site:

```
exec sp_columns [Site]
```

	TABLE_QUALIFIER	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME	PRECISION	LENGTH	SCALE
1	DB_91387_spring2018db	dbo	Site	sitPID	4	int	10	4	0
2	DB_91387_spring2018db	dbo	Site	sitName	12	varchar	50	50	NULL
3	DB_91387_spring2018db	dbo	Site	sitDescription	12	varchar	50	50	NULL

```
select * from [Site]
```

sitPID	sitName	sitDescription
1	First Site	Primary
2	Second Site	Secondary
3	Third Site	Third
4	Fourth Site	Fourth
5	Fifth Site	Fifth
6	Sixth Site	Sixth
7	Seventh Site	Seventh
8	Eighth Site	Eighth
9	Ninth Site	Ninth
10	Tenth Site	Tenth

Sponsor:

```
exec sp_columns Sponsor
```

	TABLE_QUALIFIER	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME	PRECISION	LENGTH	SCALE
1	DB_91387_spring2018db	dbo	Sponsor	spoPID	4	int	10	4	0
2	DB_91387_spring2018db	dbo	Sponsor	spoName	12	varchar	50	50	NULL
3	DB_91387_spring2018db	dbo	Sponsor	spoPhoneNumber	12	varchar	25	25	NULL

```
select * from Sponsor
```

spoPID	spoName	spoPhoneNumber
1	First Sponsor	128394857
2	Second Sponsor	1948475596
3	Third Sponsor	3886604857
4	Fourth Sponsor	2959688473
5	Fifth Sponsor	1992939945
6	Sixth Sponsor	2999394055
7	Seventh Sponsor	1002004956
8	Eighth Sponsor	1002030054
9	Ninth Sponsor	1002030056
10	Tenth Sponsor	1002035783

Doctor:

```
exec sp_columns Doctor
```

	TABLE_QUALIFIER	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME	PRECISION	LENGTH	SCALE
1	DB_91387_spring2018db	dbo	Doctor	docPID	4	int	10	4	0
2	DB_91387_spring2018db	dbo	Doctor	docName	12	varchar	100	100	NULL
3	DB_91387_spring2018db	dbo	Doctor	docSpecialization	12	varchar	50	50	NULL
4	DB_91387_spring2018db	dbo	Doctor	docPhoneNumber	12	varchar	25	25	NULL

select * from Doctor

docPID	docName	docSpecialization	docPhoneNumber
1	First Doctor	Vision	1234984102
2	Second Doctor	Olfactory	129472178
3	Third Doctor	Research	1027384957
4	Fourth Doctor	Research	1029992716
5	Fifth Doctor	Vision	7738291826
6	Sixth Doctor	Cancer	1029783821
7	Seventh Doctor	Cardiovascular	1002838477
8	Eighth Doctor	Psychology	1022283371
9	Ninth Doctor	Family Psychology	8566377459
10	Tenth Doctor	Research	1002884772

Study:

exec sp_columns Study

	TABLE_QUALIFIER	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME	PRECISION	LENGTH	SCALE	RADIX	NULLABLE	REMARKS	COLUMN_D
1	DB_91387_spring2018db	dbo	Study	stuPID	4	int	10	4	0	10	1	NULL	NULL
2	DB_91387_spring2018db	dbo	Study	stuName	12	varchar	50	50	NULL	NULL	1	NULL	NULL
3	DB_91387_spring2018db	dbo	Study	stuDescription	12	varchar	75	75	NULL	NULL	1	NULL	NULL
4	DB_91387_spring2018db	dbo	Study	stuEnrollmentGoal	4	int	10	4	0	10	1	NULL	NULL
5	DB_91387_spring2018db	dbo	Study	stuSponsorLinkID	4	int	10	4	0	10	1	NULL	NULL
6	DB_91387_spring2018db	dbo	Study	stuSiteLinkID	4	int	10	4	0	10	1	NULL	NULL

select * from Study

	stuPID	stuName	stuDescription	stuEnrollmentGoal	stuSponsorLinkID	stuSiteLinkID
1	1	First Study	Primary	25	1	4
2	2	Second Study	Second	50	4	2
3	3	Third Study	Third	100	3	8
4	4	Fourth Study	Fourth	20	4	4
5	5	Fifth Study	Fifth	10	5	5
6	6	Sixth Study	Sixth	10	6	7
7	7	Seventh Study	Seventh	15	7	8
8	8	Eighth Study	Eighth	20	7	2
9	9	Ninth Study	Ninth	50	5	6
10	10	Tenth Study	Tenth	100	4	9

StudyArm:

exec sp_columns StudyArm

	TABLE_QUALIFIER	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME	PRECISION	LENGTH	SCALE
1	DB_91387_spring2018db	dbo	StudyArm	sarPID	4	int	10	4	0
2	DB_91387_spring2018db	dbo	StudyArm	sarName	12	varchar	50	50	NULL
3	DB_91387_spring2018db	dbo	StudyArm	sarDescription	12	varchar	75	75	NULL
4	DB_91387_spring2018db	dbo	StudyArm	sarStudyLinkID	4	int	10	4	0

select * from StudyArm

sarPID	sarName	sarDescription	sarStudyLinkID
1	First Arm	Primary	1
2	Second Arm	Secondary	1
3	First Arm	Primary	2
4	Second Arm	Secondary	2
5	First Arm	Primary	3
6	Second Arm	Secondary	3
7	First Arm	Primary	4
8	Second Arm	Secondary	4
9	First Arm	Primary	5
10	Second Arm	Secondary	5
11	First Arm	Primary	6
12	Second Arm	Secondary	6
13	First Arm	Primary	7
14	Second Arm	Secondary	7
15	First Arm	Primary	8
16	Second Arm	Secondary	8
17	First Arm	Primary	9
18	Second Arm	Secondary	9
19	First Arm	Primary	10
20	Second Arm	Secondary	10

Activity:

`exec sp_columns Activity`

	TABLE_QUALIFIER	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME	PRECISION	LENGTH	SCALE
1	DB_91387_spring2018db	dbo	Activity	actPID	4	int	10	4	0
2	DB_91387_spring2018db	dbo	Activity	actName	12	varchar	50	50	NULL
3	DB_91387_spring2018db	dbo	Activity	actDescription	12	varchar	75	75	NULL
4	DB_91387_spring2018db	dbo	Activity	actDuration	4	int	10	4	0
5	DB_91387_spring2018db	dbo	Activity	actCost	6	float	15	8	NULL
6	DB_91387_spring2018db	dbo	Activity	actStudyArmLinkID	4	int	10	4	0

`select * from Activity`

actPID	actName	actDescription	actDuration	actCost	actStudyArmLinkID
1	First Activity	First	60	95	1
2	Second Activity	Second	60	100	1
3	Third Activity	Third	60	39.99	1
4	First Activity	First	60	60	2
5	Second Activity	Second	60	75	2
6	Third Activity	Third	60	150	2
7	First Activity	First	60	95	3
8	Second Activity	Second	60	100	3
9	Third Activity	Third	60	39.99	3
10	First Activity	First	60	60	4
11	Second Activity	Second	60	75	4
12	Third Activity	Third	60	150	4
13	First Activity	First	60	95	5
14	Second Activity	Second	60	100	5
15	Third Activity	Third	60	39.99	5
16	First Activity	First	60	60	6
17	Second Activity	Second	60	75	6
18	Third Activity	Third	60	150	6
19	First Activity	First	60	95	7
20	Second Activity	Second	60	100	7
21	Third Activity	Third	60	39.99	7
22	First Activity	First	60	60	8
23	Second Activity	Second	60	75	8
24	Third Activity	Third	60	150	8
25	First Activity	First	60	95	9
26	Second Activity	Second	60	100	9
27	Third Activity	Third	60	39.99	9
28	First Activity	First	60	60	10

Visit:

`exec sp_columns Visit`

	TABLE_QUALIFIER	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME	PRECISION	LENGTH	SCALE
1	DB_91387_spring2018db	dbo	Visit	visPID	4	int	10	4	0
2	DB_91387_spring2018db	dbo	Visit	visName	12	varchar	50	50	NULL
3	DB_91387_spring2018db	dbo	Visit	visDescription	12	varchar	75	75	NULL
4	DB_91387_spring2018db	dbo	Visit	visStart	11	datetime	23	16	3
5	DB_91387_spring2018db	dbo	Visit	visEnd	11	datetime	23	16	3
6	DB_91387_spring2018db	dbo	Visit	visStudyArmLinkID	4	int	10	4	0

`select * from Visit`

visPID	visName	visDescription	visStart	visEnd	visStudyArmLinkID
1	First Visit	First	2018-05-01 00:00:00.000	1900-01-01 00:00:00.000	1
2	Second Visit	Second	2018-05-02 00:00:00.000	1900-01-01 00:00:00.000	1
3	Third Visit	Third	2018-05-03 00:00:00.000	1900-01-01 00:00:00.000	1
4	First Visit	First	2018-05-01 00:00:00.000	1900-01-01 00:00:00.000	2
5	Second Visit	Second	2018-05-02 00:00:00.000	1900-01-01 00:00:00.000	2
6	Third Visit	Third	2018-05-03 00:00:00.000	1900-01-01 00:00:00.000	2
7	First Visit	First	2018-05-01 00:00:00.000	1900-01-01 00:00:00.000	3
8	Second Visit	Second	2018-05-02 00:00:00.000	1900-01-01 00:00:00.000	3
9	Third Visit	Third	2018-05-03 00:00:00.000	1900-01-01 00:00:00.000	3
10	First Visit	First	2018-05-01 00:00:00.000	1900-01-01 00:00:00.000	4
11	Second Visit	Second	2018-05-02 00:00:00.000	1900-01-01 00:00:00.000	4
12	Third Visit	Third	2018-05-03 00:00:00.000	1900-01-01 00:00:00.000	4
13	First Visit	First	2018-05-01 00:00:00.000	1900-01-01 00:00:00.000	5
14	Second Visit	Second	2018-05-02 00:00:00.000	1900-01-01 00:00:00.000	5
15	Third Visit	Third	2018-05-03 00:00:00.000	1900-01-01 00:00:00.000	5
16	First Visit	First	2018-05-01 00:00:00.000	1900-01-01 00:00:00.000	6
17	Second Visit	Second	2018-05-02 00:00:00.000	1900-01-01 00:00:00.000	6
18	Third Visit	Third	2018-05-03 00:00:00.000	1900-01-01 00:00:00.000	6
19	First Visit	First	2018-05-01 00:00:00.000	1900-01-01 00:00:00.000	7
20	Second Visit	Second	2018-05-02 00:00:00.000	1900-01-01 00:00:00.000	7
21	Third Visit	Third	2018-05-03 00:00:00.000	1900-01-01 00:00:00.000	7
22	First Visit	First	2018-05-01 00:00:00.000	1900-01-01 00:00:00.000	8
23	Second Visit	Second	2018-05-02 00:00:00.000	1900-01-01 00:00:00.000	8
24	Third Visit	Third	2018-05-03 00:00:00.000	1900-01-01 00:00:00.000	8
25	First Visit	First	2018-05-01 00:00:00.000	1900-01-01 00:00:00.000	9
26	Second Visit	Second	2018-05-02 00:00:00.000	1900-01-01 00:00:00.000	9
27	Third Visit	Third	2018-05-03 00:00:00.000	1900-01-01 00:00:00.000	9
28	First Visit	First	2018-05-01 00:00:00.000	1900-01-01 00:00:00.000	10

Appointment:

`exec sp_columns Appointment`

	TABLE_QUALIFIER	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME	PRECISION	LENGTH	SCALE
1	DB_91387_spring2018db	dbo	Appointment	appPID	4	int	10	4	0
2	DB_91387_spring2018db	dbo	Appointment	appStartTime	11	datetime	23	16	3
3	DB_91387_spring2018db	dbo	Appointment	appEndTime	11	datetime	23	16	3
4	DB_91387_spring2018db	dbo	Appointment	appCompleted	4	int	10	4	0
5	DB_91387_spring2018db	dbo	Appointment	appStudyArmLinkID	4	int	10	4	0
6	DB_91387_spring2018db	dbo	Appointment	appActivityLinkID	4	int	10	4	0
7	DB_91387_spring2018db	dbo	Appointment	appVisitLinkID	4	int	10	4	0
8	DB_91387_spring2018db	dbo	Appointment	appSubjectLinkID	4	int	10	4	0
9	DB_91387_spring2018db	dbo	Appointment	appDoctorLinkID	4	int	10	4	0

`select * from Appointment`

appPID	appStartTime	appEndTime	appCompleted	appStudyArmLinkID	appActivityLinkID	appVisitLinkID	appSubjectLinkID	appDoctorLinkID
1	2018-05-01 12:00:00.000	2018-05-01 13:00:00.000	0	1	3	1	1	1
1	2018-05-01 12:00:00.000	2018-05-01 13:00:00.000	0	1	2	2	1	1
1	2018-05-01 12:00:00.000	2018-05-01 13:00:00.000	0	1	1	3	1	1
1	2018-05-01 12:00:00.000	2018-05-01 13:00:00.000	0	2	4	6	2	2
1	2018-05-01 12:00:00.000	2018-05-01 13:00:00.000	0	2	5	5	3	2
1	2018-05-01 12:00:00.000	2018-05-01 13:00:00.000	0	2	6	4	3	2
1	2018-04-01 12:00:00.000	2018-04-01 13:00:00.000	1	3	7	9	4	1
1	2018-04-01 12:00:00.000	2018-04-01 13:00:00.000	1	3	8	8	4	1
1	2018-04-01 12:00:00.000	2018-04-01 13:00:00.000	1	3	9	7	4	1
1	2018-04-01 12:00:00.000	2018-04-01 13:00:00.000	1	4	11	11	5	3
1	2018-04-01 12:00:00.000	2018-04-01 13:00:00.000	1	4	10	10	5	3
1	2018-04-01 12:00:00.000	2018-04-01 13:00:00.000	1	4	12	12	6	3
1	2018-03-01 12:00:00.000	1900-01-01 00:00:00.000	0	5	13	14	7	4
1	2018-03-01 12:00:00.000	2018-03-01 13:00:00.000	1	5	14	13	7	4
1	2018-03-01 12:00:00.000	2018-03-01 13:00:00.000	1	5	15	14	8	5
1	2018-03-01 12:00:00.000	2018-03-01 13:00:00.000	1	4	17	16	9	6
1	2018-03-01 12:00:00.000	2018-03-01 13:00:00.000	1	4	18	17	9	7
1	2018-03-01 12:00:00.000	2018-03-01 13:00:00.000	1	4	16	18	9	8

DocumentInfo:

```
exec sp_columns DocumentInfo
```

	TABLE_QUALIFIER	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME	PRECISION	LENGTH	SCALE
1	DB_91387_spring2018db	dbo	DocumentInfo	dinPID	4	int	10	4	0
2	DB_91387_spring2018db	dbo	DocumentInfo	dinDescription	12	varchar	75	75	NULL
3	DB_91387_spring2018db	dbo	DocumentInfo	dinFileName	-1	text	2147483647	2147483647	NULL
4	DB_91387_spring2018db	dbo	DocumentInfo	dinBLOB	-1	text	2147483647	2147483647	NULL
5	DB_91387_spring2018db	dbo	DocumentInfo	dinVersion	4	int	10	4	0
6	DB_91387_spring2018db	dbo	DocumentInfo	dinStudyLinkID	4	int	10	4	0

```
select * from DocumentInfo
```

dinPID	dinDescription	dinFileName	dinBLOB	dinVersion	dinStudyLinkID
1	Patient History	somefilename.pdf	0x0001e240	2	1
2	Notes	notes.txt	0x0001e240	6	1
3	Regulatory	regdocs.pdf	0x0001e240	2	1
4	Release of Liability	rol.pdf	0x0001e240	2	1
5	Patient History	somefilename.pdf	0x0001e240	2	2
6	Notes	notes.txt	0x0001e240	6	2
7	Regulatory	regdocs.pdf	0x0001e240	2	2
8	Release of Liability	rol.pdf	0x0001e240	2	2
9	Patient History	somefilename.pdf	0x0001e240	2	3
10	Notes	notes.txt	0x0001e240	6	3
11	Regulatory	regdocs.pdf	0x0001e240	2	3
12	Release of Liability	rol.pdf	0x0001e240	2	3
13	Patient History	somefilename.pdf	0x0001e240	2	4
14	Notes	notes.txt	0x0001e240	6	4
15	Regulatory	regdocs.pdf	0x0001e240	2	4
16	Release of Liability	rol.pdf	0x0001e240	2	4
17	Patient History	somefilename.pdf	0x0001e240	2	5
18	Notes	notes.txt	0x0001e240	6	5
19	Regulatory	regdocs.pdf	0x0001e240	2	5
20	Release of Liability	rol.pdf	0x0001e240	2	5
21	Patient History	somefilename.pdf	0x0001e240	2	6
22	Notes	notes.txt	0x0001e240	6	6
23	Regulatory	regdocs.pdf	0x0001e240	2	6
24	Release of Liability	rol.pdf	0x0001e240	2	6
25	Patient History	somefilename.pdf	0x0001e240	2	7
26	Notes	notes.txt	0x0001e240	6	7
27	Regulatory	regdocs.pdf	0x0001e240	2	7
28	Release of Liability	rol.pdf	0x0001e240	2	7
29	Patient History	somefilename.pdf	0x0001e240	2	8
30	Notes	notes.txt	0x0001e240	6	8

AssignedTo:

```
exec sp_columns AssignedTo
```

TABLE_QUALIFIER	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME	PRECISION	LENGTH	SCALE
DB_91387_spring2018db	dbo	AssignedTo	atoPID	4	int	10	4	0
DB_91387_spring2018db	dbo	AssignedTo	atoStartDate	11	datetime	23	16	3
DB_91387_spring2018db	dbo	AssignedTo	atoEndDate	11	datetime	23	16	3
DB_91387_spring2018db	dbo	AssignedTo	atoLinkType	4	int	10	4	0
DB_91387_spring2018db	dbo	AssignedTo	atoSubjectLinkID	4	int	10	4	0
DB_91387_spring2018db	dbo	AssignedTo	atoDoctorLinkID	4	int	10	4	0
DB_91387_spring2018db	dbo	AssignedTo	atoStudyLinkID	4	int	10	4	0

```
select * from AssignedTo
```

atoPID	atoStartDate	atoEndDate	atoLinkType	atoSubjectLinkID	atoDoctorLinkID	atoStudyLinkID
1	2018-01-01 00:00:00.000	1900-01-01 00:00:00.000	1	1	NULL	1
2	2018-01-02 00:00:00.000	1900-01-01 00:00:00.000	1	2	NULL	1
3	2018-01-03 00:00:00.000	1900-01-01 00:00:00.000	1	3	NULL	2
4	2018-01-04 00:00:00.000	1900-01-01 00:00:00.000	1	4	NULL	2
5	2018-01-05 00:00:00.000	1900-01-01 00:00:00.000	1	5	NULL	3
6	2018-01-06 00:00:00.000	1900-01-01 00:00:00.000	1	6	NULL	3
7	2018-01-07 00:00:00.000	1900-01-01 00:00:00.000	1	7	NULL	4
8	2018-01-08 00:00:00.000	1900-01-01 00:00:00.000	1	8	NULL	4
9	2018-01-09 00:00:00.000	1900-01-01 00:00:00.000	1	9	NULL	5
10	2018-01-10 00:00:00.000	1900-01-01 00:00:00.000	1	10	NULL	5
11	2018-01-01 00:00:00.000	1900-01-01 00:00:00.000	2	NULL	1	1
12	2018-01-02 00:00:00.000	1900-01-01 00:00:00.000	2	NULL	2	1
13	2018-01-03 00:00:00.000	1900-01-01 00:00:00.000	2	NULL	3	2
14	2018-01-04 00:00:00.000	1900-01-01 00:00:00.000	2	NULL	4	2
15	2018-01-05 00:00:00.000	1900-01-01 00:00:00.000	2	NULL	5	3
16	2018-01-06 00:00:00.000	1900-01-01 00:00:00.000	2	NULL	6	3
17	2018-01-07 00:00:00.000	1900-01-01 00:00:00.000	2	NULL	7	4
18	2018-01-08 00:00:00.000	1900-01-01 00:00:00.000	2	NULL	8	4
19	2018-01-09 00:00:00.000	1900-01-01 00:00:00.000	2	NULL	9	5
20	2018-01-10 00:00:00.000	1900-01-01 00:00:00.000	2	NULL	10	5

3.4.2 Example Queries in SQL

These are extra queries designed to provide useful information to the front end application:

Query #1:

```
-- Query 1  
-- List subjects and their completed activities in 2018 before April 1st
```

```
select su.subPID, su.subInitials, ap.appEndTime, ac.actName, ac.actDuration, ac.actCost from Appointment as ap  
inner join Activity ac on ac.actPID = ap.appActivityLinkID  
inner join Subject su on su.subPID = ap.appSubjectLinkID where ap.appEndTime >= '2018-01-01'  
and ap.appEndTIme < '2018-04-01'
```

Result:

The screenshot shows a SQL query results window. At the top, there are tabs for 'Results' and 'Messages'. The 'Results' tab is selected, displaying a table with the following data:

	subPID	subInitials	appEndTime	actName	actDuration	actCost
1	7	PO	2018-03-01 13:00:00.000	Second Activity	60	100
2	8	MN	2018-03-01 13:00:00.000	Third Activity	60	39.99
3	9	CO	2018-03-01 13:00:00.000	First Activity	60	60
4	9	CO	2018-03-01 13:00:00.000	Second Activity	60	75
5	9	CO	2018-03-01 13:00:00.000	Third Activity	60	150

Below the table, a yellow bar displays the message: 'Query executed successfully.' with a checkmark icon. To the right of the bar, the text 'tcp:s' is visible.

Query #2:

```
-- Query 2  
-- List Doctors not involved in any Appointments
```

```
select d.docPID, d.docName, d.docSpecialization, d.docPhoneNumber from Doctor as d
```

```
where not exists(select * from Appointment where appDoctorLinkID = d.docPID) Result:
```

	docPID	docName	docSpecialization	docPhoneNumber
1	9	Ninth Doctor	Family Psychology	8566377459
2	10	Tenth Doctor	Research	1002884772

#3:

```
-- Query 3  
-- List all Studies and their Sponsor that have at least 3 completed Appointments  
select s.stuPID, s.stuName, s.stuEnrollmentGoal, sa.sarPID, sa.sarName, sp.spoPID, sp.spoName from Study  
as s  
inner join StudyArm as sa on sa.sarStudyLinkID = s.stuPID  
inner join Sponsor as sp  
on sp.spoPID = s.stuSponsorLinkID where exists (select appStudyArmLinkID, count(*) as  
CompletedAppointmentsByStudy  
from Appointment  
where appCompleted = 1 and appStudyArmLinkID = sa.sarPID  
group by appStudyArmLinkID  
having  
count(*) >= 3)
```

Result:

	stuPID	stuName	stuEnrollmentGoal	sarPID	sarName	spoPID	spoName
1	2	Second Study	50	3	First Arm	4	Fourth Sponsor
2	2	Second Study	50	4	Second Arm	4	Fourth Sponsor

4. Oracle Database Management System PL/SQL Components

4.1 Oracle PL/SQL

As stated before, we are using the Oracle PL/SQL DBMS, instead we are using Visual Studio along with SQL Server Management Studio. Despite this, they are very similar, and share the same structure, but for the sake of consistency the following will be in MySQL.

Just as in PL/SQL we will be using stored procedures to hide complex sections of code from the user. A stored procedure can be thought of as a function that, when called upon, will perform a particular task. We will also cover triggers which, as the name implies, will only be executed on a particular event occurrence.

4.1.1 Stored Procedures

In this section we will highlight three separate stored procedures that will perform a deletion, an insertion, and also one for ...

Deletion:

```
CREATE PROC DeleteDoctor (@docPID int) AS  
DELETE from Doctor  
WHERE docPID = @docPID  
  
exec DeleteDoctor 21
```

Insertion:

```
ALTER PROC InsertDoctor(@docName varchar(10), @docSpecialization varchar(10), @docPhoneNumber  
bigint) AS  
  
    insert [Doctor] (docName, docSpecialization, docPhoneNumber)  
    values(@docName, @docSpecialization, @docPhoneNumber)  
  
exec InsertDoctor "Dr Strange", "Brain Guy", 6618084134
```

Update:

```
CREATE PROC UpdateDoctor(@docPID int, @docName varchar(100)) AS  
    update Doctor  
    set docName = @docName  
    where docPID = @docPID  
exec UpdateDoctor 23, "Hugo Strange"
```

4.1.2 Triggers

This section will be dedicated to the triggers. We will demonstrate three triggers. They are as follows: Deletion

Deletion:

```
CREATE TRIGGER [dbo].[trig_DeleteAppointments]
    ON [dbo].[Doctor]
    FOR DELETE
AS
    DELETE FROM Appointment
    WHERE appDoctorLinkID IN(SELECT deleted.docPID FROM deleted)
GO

ALTER TABLE [dbo].[Doctor] DISABLE TRIGGER [trig_DeleteAppointments]
GO
```

4.1.3 Putting Our Triggers and Stored Procedures into Packages

Instead of putting all of our Triggers and Stored Procedure into packages we put them into a script that can be executed, and it will not only populate all information but also create the entire database. In the following section we will cover what is a package and what the syntax for it is.

4.2 Postgres PL/pgSQL

This section will include definitions of stored procedures/functions, triggers, and packages. It will also compare other DMBS systems that to the one we are using for our project. And finally it will include some screenshots of our stored procedures and functions.

In PL/pgSQL a stored procedure is a set of procedural statements that are stored within the database. These statements can be invoked from the interface. In our case we are using MS Visual Studio and SQL Server Management Studio so our procedures and triggers will be in that format. They are very similar and do not differ very much from the others.

Stored Procedure: As defined above, a stored procedure is a set of procedural statements that may be invoked from the interface. The syntax for the creation of one is as follows:

```
CREATE {or ALTER} PROC name_of_procedure AS
    SELECT      name_of_items_wanted
    FROM        name_of_table
```

WHERE *condition_if_needed*

Stored Function: A stored function is very similar to a stored procedure. The difference is that a function will return. The reason behind this is because the function will calculate a value and return the value, without having to modify the data within the database.

CREATE {or **ALTER**} **PROC** *name_of_procedure AS*

SELECT *name of items wanted*
FROM *name of table*
WHERE *condition if needed*

RETURN *value to be returned*

Packages: In terms of MySQL, there is no equivalent for packages. Due to this, we will demonstrate packages in the syntax of PL/pgSQL. A package is a combination of stored procedures, stored functions, types, and other objects into one unit. The package will contain all the prototypes of the all that is included. In the body, the definitions of all these are included. Here is some sample syntax:

CREATE {or **REPLACE**} **PACKAGE** *name of package AS*
 all that is wanted within the package
END

CREATE {or **REPLACE**} **PACKAGE body** *name of package AS*
 all that is wanted within the package
END

Triggers: A trigger is simply a stored procedure that is “trigger” or engaged in response to a certain action. These can be activated before, during, or after the event has happened. They may even happen instead of an event that was suppose to take place. Here is some sample syntax:

Example 1:

CREATE TRIGGER *name of trigger BEFORE INSERT ON* *columns name*
FOR EACH ROW *trigger body, what we want it to do*

Example 2:

```

CREATE TRIGGER name of trigger BEFORE INSERT ON columns name
FOR EACH ROW trigger body, what we want it to do
BEGIN this could also be used to add more function to trigger

END

```

It is also important to note that where **INSERT** is seen, it can be replaced with **UPDATE** or **DELETE**. Where **BEFORE** can be seen, it can be replaced with **AFTER**. This is all done with respect to what we want to accomplish.

4.3 Postgres/pgSQL Subprograms

In this section we will demonstrate the three stored procedures/functions and also the three triggers. The code will come from section 4.1. Here we will simply display the results of what that code does when it is executed in our database.

InsertDoctorProc: As can be seen below, this is the table Doctor before I call the Insert procedure.

Before:

	docPID	docName	docSpecialization	docPhoneNumber
1	1	Leonard Bones McCoy	Vision	(123)-498-4102
2	2	Beverly Crusher	Olfactory	(122)-947-2178
3	3	Ogawa	Research	(102)-738-4957
4	4	Katherine Pulaski	Research	(102)-999-2716
5	5	Julian Bashir	Vision	(773)-829-1826
6	6	Phlox	Cancer	(102)-978-3821
7	7	Jadzia Dax	Cardiovascular	(100)-283-8477
8	8	Vic Fontaine	Psychology	(102)-228-3371
9	9	Kathryn Janeway	Family Psychology	(856)-637-7459
10	10	Geordi La Forge	Research	(100)-288-4772

Query executed successfully.

After:

100 %

	Results	Messages		
	docPID	docName	docSpecialization	docPhoneNumber
1	1	Leonard Bones McCoy	Vision	(123)-498-4102
2	2	Beverly Crusher	Olfactory	(122)-947-2178
3	3	Ogawa	Research	(102)-738-4957
4	4	Katherine Pulaski	Research	(102)-999-2716
5	5	Julian Bashir	Vision	(773)-829-1826
6	6	Phlox	Cancer	(102)-978-3821
7	7	Jadzia Dax	Cardiovascular	(100)-283-8477
8	8	Vic Fontaine	Psychology	(102)-228-3371
9	9	Kathryn Janeway	Family Psychology	(856)-637-7459
10	10	Geordi La Forge	Research	(100)-288-4772
11	21	Dr Strange	Brain Guy	6618084134

Query executed successfully.

As can be seen the result of the call to the procedure result in the addition of Dr Strange.

DeleteDoctorProc: The following will be the result of removing a doctor from the table.

Before:

100 %

	Results	Messages		
	docPID	docName	docSpecialization	docPhoneNumber
1	1	Leonard Bones McCoy	Vision	(123)-498-4102
2	2	Beverly Crusher	Olfactory	(122)-947-2178
3	3	Ogawa	Research	(102)-738-4957
4	4	Katherine Pulaski	Research	(102)-999-2716
5	5	Julian Bashir	Vision	(773)-829-1826
6	6	Phlox	Cancer	(102)-978-3821
7	7	Jadzia Dax	Cardiovascular	(100)-283-8477
8	8	Vic Fontaine	Psychology	(102)-228-3371
9	9	Kathryn Janeway	Family Psychology	(856)-637-7459
10	10	Geordi La Forge	Research	(100)-288-4772
11	22	Dr Strange	Brain Guy	6618084134

Query executed successfully.

After:

The screenshot shows a SQL Server Management Studio window with the 'Results' tab selected. The results grid displays a table with four columns: docPID, docName, docSpecialization, and docPhoneNumber. The data consists of ten rows, each representing a doctor's information. The first row has docPID 1, docName 'Leonard Bones McCoy', docSpecialization 'Vision', and docPhoneNumber '(123)-498-4102'. The last row has docPID 10, docName 'Geordi La Forge', docSpecialization 'Research', and docPhoneNumber '(100)-288-4772'. A message bar at the bottom of the results pane states 'Query executed successfully.'

	docPID	docName	docSpecialization	docPhoneNumber
1	1	Leonard Bones McCoy	Vision	(123)-498-4102
2	2	Beverly Crusher	Olfactory	(122)-947-2178
3	3	Ogawa	Research	(102)-738-4957
4	4	Katherine Pulaski	Research	(102)-999-2716
5	5	Julian Bashir	Vision	(773)-829-1826
6	6	Phlox	Cancer	(102)-978-3821
7	7	Jadzia Dax	Cardiovascular	(100)-283-8477
8	8	Vic Fontaine	Psychology	(102)-228-3371
9	9	Kathryn Janeway	Family Psychology	(856)-637-7459
10	10	Geordi La Forge	Research	(100)-288-4772

As can be seen, Dr Strange is no longer part of the table.

UpdateDoctorProc: To demonstrate this UPDATE procedure, I will already have Dr Strange in the table and I will change his name to Hugo Strange.

Before:

The screenshot shows a SQL Server Management Studio window with the 'Results' tab selected. The results grid displays a table with four columns: docPID, docName, docSpecialization, and docPhoneNumber. The data consists of eleven rows, each representing a doctor's information. The first ten rows are identical to the ones in the previous screenshot. The last row has docPID 11, docName 'Dr Strange', docSpecialization 'Brain Guy', and docPhoneNumber '6618084134'. A message bar at the bottom of the results pane states 'Query executed successfully.'

	docPID	docName	docSpecialization	docPhoneNumber
1	1	Leonard Bones McCoy	Vision	(123)-498-4102
2	2	Beverly Crusher	Olfactory	(122)-947-2178
3	3	Ogawa	Research	(102)-738-4957
4	4	Katherine Pulaski	Research	(102)-999-2716
5	5	Julian Bashir	Vision	(773)-829-1826
6	6	Phlox	Cancer	(102)-978-3821
7	7	Jadzia Dax	Cardiovascular	(100)-283-8477
8	8	Vic Fontaine	Psychology	(102)-228-3371
9	9	Kathryn Janeway	Family Psychology	(856)-637-7459
10	10	Geordi La Forge	Research	(100)-288-4772
11	23	Dr Strange	Brain Guy	6618084134

After:

	docPID	docName	docSpecialization	docPhoneNumber
1	1	Leonard Bones McCoy	Vision	(123)-498-4102
2	2	Beverly Crusher	Olfactory	(122)-947-2178
3	3	Ogawa	Research	(102)-738-4957
4	4	Katherine Pulaski	Research	(102)-999-2716
5	5	Julian Bashir	Vision	(773)-829-1826
6	6	Phlox	Cancer	(102)-978-3821
7	7	Jadzia Dax	Cardiovascular	(100)-283-8477
8	8	Vic Fontaine	Psychology	(102)-228-3371
9	9	Kathryn Janeway	Family Psychology	(856)-637-7459
10	10	Geordi La Forge	Research	(100)-288-4772
11	23	Hugo Strange	Brain Guy	6618084134

Query executed successfully.

As can be seen, Dr Strange is now Hugo Strange. The docPID is still the same to confirm that it is the same doctor, just with an update name.

Delete Trigger: This trigger would delete all appointments related to a doctor that was deleted.

Before

```
select appPID  
from Appointment  
where appDoctorLinkID = 10
```

100 %

Results Messages

	appPID
1	64
2	65
3	66
4	67
5	68
6	69
7	70

Query executed successfully.

After

```
select appPID  
from Appointment  
where appDoctorLinkID = 10  
  
delete Doctor  
where docPID = 10
```

100 %

Results Messages

appPID

Query executed successfully.

The trigger was meant to delete all appointments that the deleted doctor was assigned too. So as can be seen above the doctor with PID 11 was assigned to appointments 64 to 70. After the doctor was deleted those appointments were also deleted.

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

This section will cover other types of popular commercial DBMS software. They are all different but at their core they are similar because the goal does not actually change very much. We will compare Oracle PL/SQL, MSS T-SQL, and also MySQL based on syntax and their functionality.

MySQL: MySQL is very similar to the other popular DBMS software but it does lack some of the functions that the others do have. For example, MySQL does not support packages. It does not offer packages for namespace like other DBMS software does. It also does not have loops. It does have *while* loops and other

basic loops, but that is it. Another thing to note is that MySQL parameters do work in much the same way as they do in PL/SQL.

Stored procedure:

```
DELIMITER //
CREATE PROCEDURE name of procedure
(IN con CHAR(20))
BEGIN

    SELECT what we want FROM column
    WHERE Continent = con;

END //
DELIMITER ;
```

Stored Function:

```
DELIMITER //
CREATE PROCEDURE name of procedure
(IN con CHAR(20))
RETURN type
BEGIN

    SELECT what we want FROM column
    WHERE Continent = con;

END //
DELIMITER ;
```

While loop:

```
[begin label:] WHILE condition DO
    Statement list
END WHILE [end label]
```

One last thing to note is the use of **DELIMITER**. This is used to change the traditional end line character of semi-colon(;) to //.

Microsoft SQL Server: T-SQL : T-SQL is probably the most unique out of the ones we will be comparing. T-SQL gives the option to encrypt the text of any procedure by simply making use of the with clause. This is useful when trying to restrict access to certain users. T-SQL also allows for the use of multiple *try-catch* blocks in a procedure. This is not possible in PL/SQL and MySQL. T-SQL functions can also return both tables and scalars. This is something that is not possible in MySQL. When it comes to parameters, all passed parameters must be preceded by an '@' character. T-SQL does have its disadvantages. Just like MySQL, T-SQL does not offer the use of *for* loops. Instead only the most basic loops are allowed to be used.

Stored Procedure:

```
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 ]

```

Stored Function:

```

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
[ ; ]

```

Basic Loop:

```

WHILE @count < count total
BEGIN
    {...statements...}
    SET @count = @count + 1
    []
END []

```

Oracle PL/SQL: This is the DBMS that we mainly focused on in class. It is also the one we used to reference for the creation of this document. This DBMS allows for the use of packages and also for the use of the *for* loop. Both of these two things are not a part of the previously mentioned DBMS software.

Stored Procedure:

```

CREATE or {REPLACE} PROCEDURE procedure name
BEGIN
    PL/SQL statements
END

```

Stored Function:

```
CREATE or {REPLACE} FUNCTION function name
  RETURN type
BEGIN
  PL/SQL statements
END
```

The loops:

```
WHILE condition
LOOP
  Statements
END LOOP;
```

```
FOR counter in range
LOOP
  Statements
END LOOP;
```

5. Graphical User Interface Design and Implementation

In this section we will cover what we did for the Graphical User Interface. We will cover how we went about creating the GUI. We will showcase some of the code we used as well as screenshots of the screens the user will encounter and also explanation as to what buttons do what. At the very end of the section we will answer some Survey questions provided by Dr.Wang.

5.1 Functionalities and User Group of the GUI Application

This section will cover the basics of the GUI and also showcase some of the code that went into writing this GUI. It will include screen shots and some samples of code that will be given some explanation.

5.1.1 Itemized Descriptions of GUI Application

By the time we had to create different User Interfaces for our user groups, we had already begin work together. We did what was instructed and we split the work in terms of the who gets to see what. We had a couple different users that would log in and interact with the database. The level at which they interact with the database varied depending on their access level. The user types are as follows:

Super User: The super user has access to everything that can be seen in the GUI. They also have the ability to add and remove users. To also remove their privileges.

Administrator: An administrator is assigned to a particular site and is the person in charge or overseer of that particular site. They have the ability to change some aspects of the site.

Coordinator: A coordinator is assigned to a particular study within a site by the Administrator of that site. From there they have access to the study but nothing beyond that study.

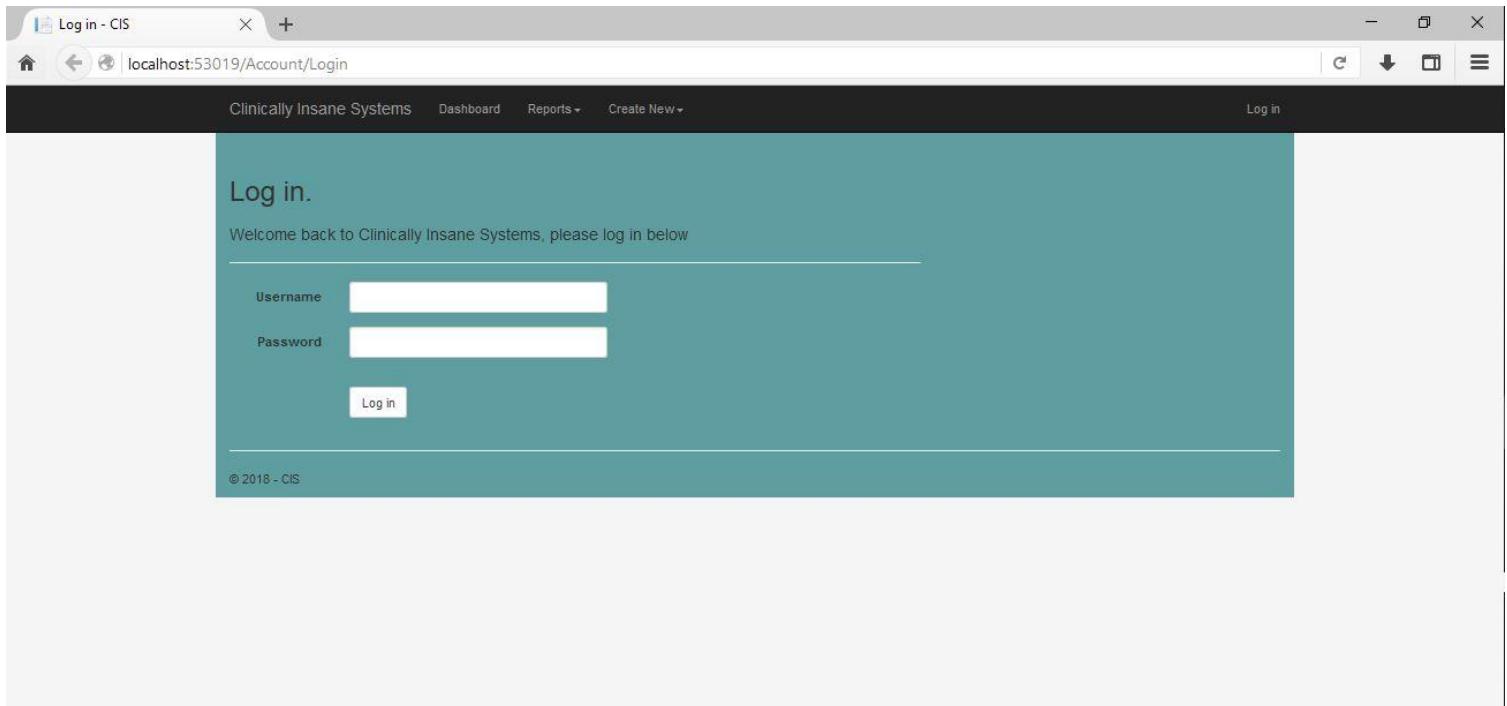
Monitor: The monitor is also assigned to a particular study by the Administrator. However, the monitor does not have the ability to make any changes to the study. Their only job is to make sure that those involved are actually doing their jobs.

For this I was put in charge of the Coordinator and the Monitor views. But as can be seen above, they do not have very much functionality. So, while I did work on these two user types, I also worked on other parts of the GUI. Our GUI was very connected that we split the work to make the best possible GUI we could, without one doing all the work.

5.1.2 Screen Shots of Application

In this section I will show some screenshots of what I worked on during this long process.

The Login



A screenshot of a web browser displaying the 'Log in - CIS' page. The URL in the address bar is 'localhost:53019/Account/Login'. The page has a dark header with the text 'Clinically Insane Systems', 'Dashboard', 'Reports', 'Create New', and 'Log in'. The main content area has a teal background and displays the text 'Log in.' and 'Welcome back to Clinically Insane Systems, please log in below'. It contains two input fields for 'Username' and 'Password', and a 'Log in' button. At the bottom, there is a copyright notice '© 2018 - CIS'.

This is what the login page looks like at the time of writing this. This is where the user type is determined. Depending on the user the GUI will change accordingly to show or hide some of the functionality. For the sake of throughout explanation, I will demonstrate what the Super User can see. And along the way, I will specify what the monitor or Coordinator would be able to see and do.

The Dashboard

The screenshot shows a web browser window titled "Hello, Cesar - CIS" with the URL "localhost:53019". The page header includes "Clinically Insane Systems :1", "Dashboard", "Reports", "Create New", and "Logout". The main content is a table titled "Studies Overview" with the following data:

Site	Study	Start	End	Arm	Enrollment Goal
Beaumont Emergency Care	Heart Monitor x1004854	05/01/2018	Incomplete	Main	25
		05/01/2018	Incomplete	Placebo Group	25
Bakersfield Clinic	Blood Pressure x1230201	04/01/2018	05/01/2018	Main	25
Beaumont Emergency Care	Arthritis Medication x2291943	02/01/2018	Incomplete	Main	15
		02/01/2018	Incomplete	Placebo Group	45
Pew Research Center	Caffeine Addiction x1204212	02/01/2018	Incomplete	Main	30
		02/01/2018	Incomplete	Placebo Group	45
Bakersfield Clinic	Shock Therapy x295848	06/01/2017	Incomplete	Main	15
Yale Campus Clinic	Torture Methods x1958382	06/01/2017	Incomplete	Main	25
		06/01/2017	Incomplete	Placebo Group	35

© 2018 - CIS

This is the Dashboard that the all users but the monitor see. The Coordinator would also be able to see this page, but it would only display the site that they are involved in. Thus having access to the study details. As the superuser, as in this case, they are able to see all sites and also all studies along with their respective arms.

Study Details

The screenshot shows a web-based application titled "Study Report for ID - CIS". The URL in the address bar is "localhost:53019/Home/StudyReportByID/1". The page has a header with "Clinically Insane Systems :1", "Dashboard", "Reports", "Create New", and "Logout" buttons. Below the header is a section titled "StudyReport with a given ID". This section contains two tables: "Study Report For Specific ID" and "Subject(s)".

Study Report For Specific ID

Study Name	Study Start Date	Study End Date	Study Arm Name	Total Cost
Heart Monitor x1004854	5/1/2018 12:00:00 AM	1/1/1900 12:00:00 AM	Main	\$1124.94
Heart Monitor x1004854	5/1/2018 12:00:00 AM	1/1/1900 12:00:00 AM	Placebo Group	\$630

Subject(s)

Initials	Date of Birth	Gender	Height	Blood Type	Status	Survival Status
RP	4/20/1992 12:00:00 AM	Male	58	O	1	Alive
AP	5/13/1994 12:00:00 AM	Male	66	A+	1	Alive
SY	8/12/2000 12:00:00 AM	Female	65	B-	1	Alive
BS	7/7/1984 12:00:00 AM	Male	90	B-	1	Alive
AP	5/13/1994 12:00:00 AM	Male	66	A+	1	Alive
SY	8/12/2000 12:00:00 AM	Female	65	B-	1	Alive
BS	7/7/1984 12:00:00 AM	Male	90	B-	1	Alive
CA	9/2/1977 12:00:00 AM	Female	29	O	1	Alive
IA	5/19/1995 12:00:00 AM	Male	39	A-	1	Alive

Once the Coordinator has selected their respective study, this is the information that is shown. They are given a status on what is going on within that study. From this page they can see the subjects that are part of the study as well as the doctors. They can also see the study arms that are part of the study. Typically these arm are name “Main” and “Placebo”.

Study Arm Details

The screenshot shows the 'StudyArmDetailsView' page from the 'Clinically Insane Systems' application. At the top, there's a navigation bar with links for 'Dashboard', 'Reports', 'Create New', and 'Logout'. Below the header, the main content area is titled 'StudyArmDetailsView'.

Study Arm Details:

Name	Duration	Visit #11	Visit #12	Visit #13
Blood Draw	60	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Random Transfusion	60	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Physical	60	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Subject Information:

Initials	Date of Birth	Gender	Height	Blood Type	Status	Survival Status
AP	5/13/1994 12:00:00 AM	Male	56	A+	1	Alive
SY	8/12/2000 12:00:00 AM	Female	65	B-	1	Alive
BS	1/7/1984 12:00:00 AM	Male	90	B-	1	Alive
CA	9/2/1977 12:00:00 AM	Female	29	O	1	Alive
IA	5/19/1995 12:00:00 AM	Male	39	A-	1	Alive

Add Subjects

Doctor Information:

Name	Specialization	Phone Number
Beverly Crusher	Olfactory	(122)-947-2178

Add Doctors

Files:

Description	File Name	Version	Study Arm Link
Patient History	somename.pdf	2	2
Notes	notes.txt	6	2
Regulatory	regdocs.pdf	2	2
Release of Liability	rol.pdf	2	2

Add Files

Buttons:

- [View Subject Report](#)
- [View Doctor Report](#)

© 2018 - CIS

This page can be seen by both the monitor and the coordinator. This page displays more information but specific to the arm. Whether it be the main arm or the placebo arm, this page displays doctors involved, subjects involved, upcoming activities, and also some buttons to show some of the reports for doctor and subject. It also shows some buttons to add doctors and subjects. In the lower right there is a table that shows the documents that are associated with the study arm. It also gives the option to add some documents.

Study Report

The screenshot shows a web-based application titled "Study Report - CIS". The URL in the address bar is "localhost:53019/Home/StudyReport". The top navigation bar includes links for "Dashboard", "Reports", "Create New", and "Logout". Below the header, the title "StudyReport" is displayed, followed by a subtitle "Study Revenue". A table titled "Study Name" lists various studies with their start and end dates, study arm names, total costs, and completion percentages. The table includes rows for "Heart Monitor x1004854", "Blood Pressure x1230201", "Medicinal Cannabis x1249382", "Arthritis Medication x2291943", "Caffeine Addiction x1204212", "Tobacco Addiction x1485293", "Weight Loss Drugs x286736", "Shock Therapy x295848", "Torture Methods x1958382", and "Lethal Injection x958381". The footer of the page contains the copyright notice "© 2018 - CIS".

Study Name	Study Start Date	Study End Date	Study Arm Name	Total Cost	Completed Apps/Total Apps
Heart Monitor x1004854	5/1/2018 12:00:00 AM	1/1/1900 12:00:00 AM	Main	\$1124.94	7 / 15
	5/1/2018 12:00:00 AM	1/1/1900 12:00:00 AM	Placebo Group	\$630	3 / 7
Blood Pressure x1230201	4/1/2018 12:00:00 AM	5/1/2018 12:00:00 AM	Main	\$469.98	4 / 6
Medicinal Cannabis x1249382	3/1/2018 12:00:00 AM	1/1/1900 12:00:00 AM	Main	\$570	3 / 6
	3/1/2018 12:00:00 AM	1/1/1900 12:00:00 AM	Placebo Group	\$469.98	4 / 6
Arthritis Medication x2291943	2/1/2018 12:00:00 AM	1/1/1900 12:00:00 AM	Main	\$570	1 / 6
Caffeine Addiction x1204212	2/1/2018 12:00:00 AM	1/1/1900 12:00:00 AM	Main	\$469.98	2 / 6
	2/1/2018 12:00:00 AM	1/1/1900 12:00:00 AM	Placebo Group	\$570	4 / 6
Tobacco Addiction x1485293	10/1/2017 12:00:00 AM	1/1/1900 12:00:00 AM	Main	\$469.98	3 / 6
Weight Loss Drugs x286736	11/1/2017 12:00:00 AM	12/1/2017 12:00:00 AM	Main	\$570	2 / 6
	11/1/2017 12:00:00 AM	12/1/2017 12:00:00 AM	Placebo Group	\$0	0 / 0
Shock Therapy x295848	6/1/2017 12:00:00 AM	1/1/1900 12:00:00 AM	Main	\$0	0 / 0
Torture Methods x1958382	6/1/2017 12:00:00 AM	1/1/1900 12:00:00 AM	Main	\$0	0 / 0
	6/1/2017 12:00:00 AM	1/1/1900 12:00:00 AM	Placebo Group	\$0	0 / 0
Lethal Injection x958381	8/1/2017 12:00:00 AM	1/1/1900 12:00:00 AM	Main	\$0	0 / 0

This is a view of a report that shows some information on all studies across all sites. This page would not be visible to the monitor or the coordinator but it is something that I worked on and wanted to showcase. The page shows how much each arm is costing and the amount of appointments each arm has. It also shows the number of appointments completed as compared to those not completed.

Appointment Statistics Report

The screenshot shows a web browser window titled "Appointment Statistics - CIS". The URL is "localhost:53019/Home/AppointmentStatistics". The page header includes "Clinically Insane Systems :1", "Dashboard", "Reports", "Create New", and "Logout". The main content area is titled "Appointment Statistics" and contains a table with the following data:

Study	Study Arm	Completed Status	Activity	Cost
Heart Monitor x1004854	Main	0	Blood Draw	\$95
	Main	0	Blood Draw	\$80
	Main	0	Physical	\$150
	Main	0	Random Transfusion	\$100
	Main	1	Blood Draw	\$95
	Main	1	Physical	\$39.99
	Placebo Group	0	Blood Draw	\$60
	Placebo Group	0	Blood Draw	\$85
	Placebo Group	0	Random Transfusion	\$100
	Placebo Group	0	Random Transfusion	\$100
Blood Pressure x1230201	Placebo Group	0	Random Transfusion	\$75
	Main	0	Physical	\$150
	Main	0	Physical	\$39.99
	Main	0	Random Transfusion	\$75
	Main	1	Physical	\$39.99
Medicinal Cannabis x1249382	Main	1	Random Transfusion	\$100
	Main	0	Blood Draw	\$60
	Main	0	EKG	\$100
	Main	0	Physical	\$39.99

This is another page that I worked on and is not visible to the monitor or the coordinator. This is a statistic page for every appointment that is scheduled for each study and each arm. This report includes those appointments that have been completed and those that have not. It shows the activity associated with the appointment and it also shows the cost of each of those activities.

Add Subject

The screenshot shows a web browser window titled "Add Subject - CIS". The URL is "localhost:53019/Home/AddSubject". The page header includes "Clinically Insane Systems :1", "Dashboard", "Reports", "Create New", and "Logout". The main content area is titled "Add Subject" and "Subject Information". It contains fields for "Initials", "Date of Birth", "Gender", "Height", "Blood Type", and "Survival Status", each with a corresponding input box. Below these fields is a "Create" button. At the bottom left is a "Back to List" link, and at the bottom right is the copyright notice "© 2018 - CIS".

This page is to add a subject to system. As can be seen above, the fields are the basic data that makes up the Subject table. Once valid data is entered and the create button is pressed, the subject is now a part of the system. This is a function that is not available to the monitor but it is available to the coordinator.

Add Site

The screenshot shows a web browser window titled "AddSite - CIS". The URL is "localhost:53019/Home/AddSite". The page header includes "Clinically Insane Systems :1", "Dashboard", "Reports", "Create New", and "Logout". The main content area is titled "Add Site" and "Site Information". It contains fields for "Name" and "Description", each with a corresponding input box. Below these fields is a "Create" button. At the bottom left is a "Back to List" link, and at the bottom right is the copyright notice "© 2018 - CIS".

This page is very basic and its only purpose is to add a site to the system. This purpose behind this is for the future creation of new studies. If they require a site that is not in the system then they can be added here. This function is NOT available to the monitor or the coordinator.

5.2 Programming Sections

In this section I will cover more of the specific code and provide some explanation for the pages that are found in the previous section. I will select some of the pages and go into details on how they were created. In addition, I will cover how we connected to the database using a connection string and also some more code that I think should be seen. Finally I will quickly cover the client side programming which is essentially the front end and all its components.

5.2.1 Server-Side Programming

For this section I will show some code from the project. Including some views and stored procedures. Because of the large amount of views and stored procedures, I will only show three of the ones that took the most thought. Since I see that this section is similar to 5.2.1 I will demonstrate these shots of code in that section. Here I will show some of the tables that we get in our GUI along with the stored procedure and view that makes it up.

This following section of code will be the for the Study Report that was shown in section 5.1.2:

View:

```
CREATE VIEW [dbo].[v_StudyReport]
AS SELECT stuPID, sarPID, stuName, stuStartDate, stuEndDate, sarName, ISNULL(sum(appCompletedStatus), 0) AS
NumberCompleted, ISNULL(SUM(actCost), 0.0) AS TotalCost, count(appPID) as TotalApp
FROM Appointment
    right outer join Activity on actPID = appActivityLinkID
    right outer join StudyArm on sarPID = actStudyArmLinkID
    right outer join Study on stuPID = sarStudyLinkID
GROUP BY stuPID, sarPID, stuName, sarName, stuStartDate, stuEndDate
GO
```

Stored Procedure:

```
CREATE PROC [dbo].[sp_DoctorReport](@docPID int) AS
Select *
FROM v_DoctorReport
WHERE docPID = @docPID
```

Code in Visual Studio

```
public ActionResult StudyReport(int? id)
{
    var sqlStuff = new sqlStuff();
    var data = sqlStuff.StudyReport();

    var model = new List<StudyReportViewModel>();
    model = ConvertDataTable<StudyReportViewModel>(data);

    return View(model);
}
```

```
static string GetAppointmentStatistics = "sp_AppointmentStatistics";
static string GetDoctorReport = "sp_DoctorReport";
static string GetSubjectReport = "sp_SubjectReport";
static string GetStudyReportByID = "sp_StudyReportByID";
static string GetStudyReport = "sp_StudyReport";
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5
6  namespace Bryan_GUI.Models
7  {
8      public class DoctorReportViewModel
9      {
10          public DoctorReportViewModel()
11          {
12
13              appPID = 0;
14              docPID = 0;
15              docName = "";
16              stuName = "";
17              sarName = "";
18              appCompletedStatus = 0;
19          }
20
21          public int appPID { get; set; }
22          public int docPID { get; set; }
23          public string docName { get; set; }
24          public string stuName { get; set; }
25          public string sarName { get; set; }
26          public int appCompletedStatus { get; set; }
27      }
28  }
```

These pieces of code will combine together to give the view that can be seen in section 5.2.1. The first piece is the view which takes appointment and joins the section from Activity, StudyArm, and Study. After that we go to the stored procedure that just selects all from the view. From there we jump into Visual studio into what is know as the home controller which creates the action for the view. Then we go into another file in our visual project which links the stored procedure to Visual Studio. The “GetDoctorReport” has the name of the stored procedure that was created in SSMS.

The follow section of code will be about the page Appointment Statistics that can be seen in section 5.2.1.

View:

```
ALTER VIEW [dbo].[v_AppointmentStatistics] AS
SELECT appCompletedStatus, stuName, sarName, appPID, stuPID, sarPID, actName, actCost
FROM Appointment
    inner join StudyArm on appStudyArmLinkID = sarPID
    inner join Study on sarStudyLinkID = stuPID
    inner join Activity on appActivityLinkID = actPID
```

Stored Procedure:

```
ALTER PROC [dbo].[sp_AppointmentStatistics]
AS
```

```

SELECT      stuName, sarName, appCompletedStatus, actName, actCost
FROM        v_AppointmentStatistics
order       by stuPID, sarPID, appCompletedStatus, actName

```

Code from Visual Studio:

```

public DataTable AppointmentStatistics()
{
    var returnValue = new DataTable();
    using (var sqlConnection = GetControlConnection())
    {
        using (var sqlCommand = new SqlCommand(GetAppointmentStatistics, sqlConnection))
        {
            sqlCommand.CommandType = CommandType.StoredProcedure;
            //sqlCommand.CommandTimeout = 60; // one minute timeout
            sqlConnection.Open();

            using (var sqlDataAdapter = new SqlDataAdapter(sqlCommand))
            {
                sqlDataAdapter.Fill(returnValue);
            }
        }
    }
    return returnValue;
}

```

```

static string GetAppointmentStatistics = "sp_AppointmentStatistics";
static string GetDoctorReport = "sp_DoctorReport";
static string GetSubjectReport = "sp_SubjectReport";
static string GetStudyReportByID = "sp_StudyReportByID";
static string GetStudyReport = "sp_StudyReport";

```

```

1  Using System;
2  Using System.Collections.Generic;
3  Using System.Linq;
4  Using System.Web;
5
6  Namespace Bryan_GUI.Models
7  {
8      public class AppointmentStatisticsViewModel
9      {
10         public AppointmentStatisticsViewModel()
11         {
12             stuName = "";
13             sarName = "";
14             appCompletedStatus = 0;
15             actName = "";
16             actCost = 0.0;
17         }
18
19         public string stuName { get; set; }
20         public string sarName { get; set; }
21         public int appCompletedStatus { get; set; }
22         public string actName { get; set; }
23         public double actCost { get; set; }
24     }
25 }

```

The code above shows how I went about creating the Appointment Statistics Report. First I created the view which was very simple and the next step was also simple, just a select all from the previously created view. After that I jump into Visual Studio to

create a view model that will receive the data from the data base and give them variables name that match those that they have. Next, like previously, I created the link to the database stored procedure than created an action link to it.

The last section I will demonstrate is another page that can be seen above the section 5.2.1.

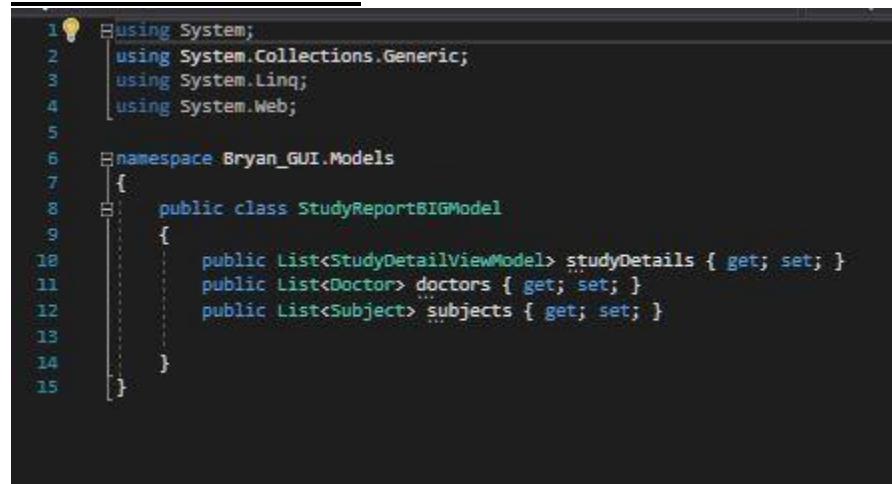
View:

```
ALTER VIEW [dbo].[v_AppointmentStatistics] AS
SELECT appCompletedStatus, stuName, sarName, appPID, stuPID, sarPID, actName, actCost
FROM Appointment
    inner join StudyArm on appStudyArmLinkID = sarPID
    inner join Study on sarStudyLinkID = stuPID
    inner join Activity on appActivityLinkID = actPID
```

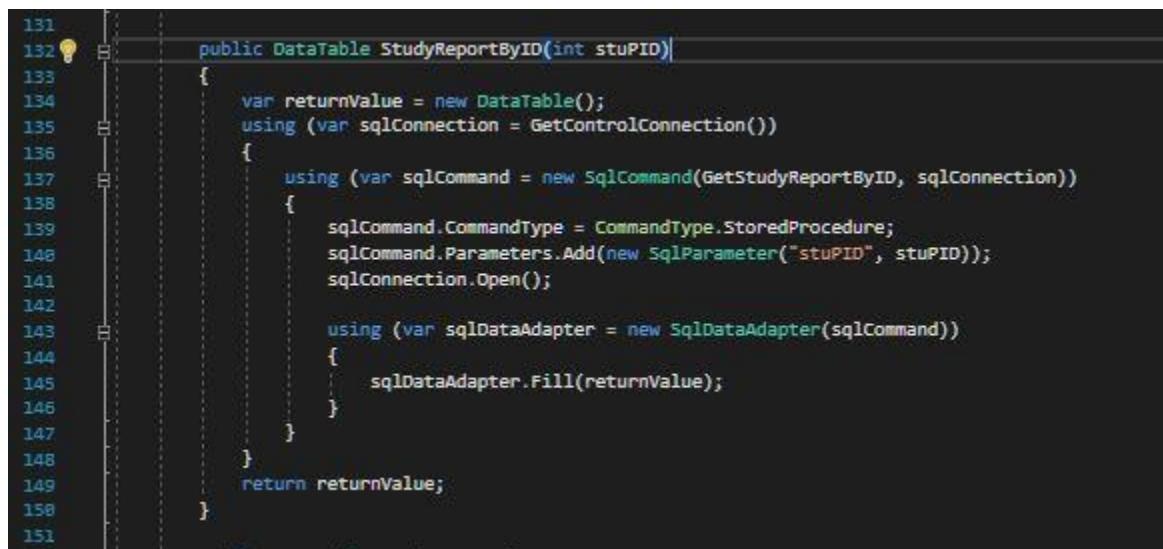
Stored Procedure:

```
ALTER proc [dbo].[sp_StudyReportByID](@stuPID int) as
select *
from v_StudyReport
where stuPID = @stuPID
```

Code from Visual Studio:



```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5
6  namespace Bryan_GUI.Models
7  {
8      public class StudyReportBIGModel
9      {
10          public List<StudyDetailViewModel> studyDetails { get; set; }
11          public List<Doctor> doctors { get; set; }
12          public List<Subject> subjects { get; set; }
13      }
14  }
```



```
131
132     public DataTable StudyReportByID(int stuPID)
133     {
134         var returnValue = new DataTable();
135         using (var sqlConnection = GetControlConnection())
136         {
137             using (var sqlCommand = new SqlCommand(GetStudyReportByID, sqlConnection))
138             {
139                 sqlCommand.CommandType = CommandType.StoredProcedure;
140                 sqlCommand.Parameters.Add(new SqlParameter("stuPID", stuPID));
141                 sqlConnection.Open();
142
143                 using (var sqlDataAdapter = new SqlDataAdapter(sqlCommand))
144                 {
145                     sqlDataAdapter.Fill(returnValue);
146                 }
147             }
148         }
149         return returnValue;
150     }
151 }
```

```
static string GetAppointmentStatistics = "sp_AppointmentStatistics";
static string GetDoctorReport = "sp_DoctorReport";
static string GetSubjectReport = "sp_SubjectReport";
static string GetStudyReportByID = "sp_StudyReportByID";
static string GetStudyReport = "sp_StudyReport";
```

```
38
39     static string GetDoctorsByStudyArm = "sp_DoctorsByStudyArm";
40     static string GetSubjectsByStudyArm = "sp_SubjectsByStudyArm";
41     static string GetDocumentsByStudyArm = "sp_DocumentsByStudyArm";
42     static string GetVisitsByStudyArm = "sp_VisitsByStudyArm";
43     static string GetActivitiesByStudyArm = "sp_ActivitiesByStudyArm";
44     static string GetAppointmentsByStudyArm = "sp_AppointmentsByStudyArm";
```

Here is the code for the Study Report page that includes the information of the study. This includes the arms, basic doctor information, and basic subject information. First to do this I had to create big model that would enclose study details, doctors, and subjects. The reason behind this is because this was the only way I could find to show multiple models in one page. Once the big model was created the rest was simple. I made the link to the database with the “Get” strings in the file. Then after that I made the action link, then the cshtml page to actually show the information in the page.

5.2.2 Middle-Tire Programming

This section is to show how we connected to the database. We did this with a connection string, which can be seen below:

```
51 <connectionStrings>
52   <add name="DefaultConnection" connectionString="Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\sspnet-Bryan_GUI-20180405013351.mdf;Initial Catalog=sspnet-Bryan_GUI-20180405013351;Integrated Security=True" providerName="System.Data.SqlClient"/>
53   <add name="SiteContext" connectionString="metadata=res://*/Models.Bryan_GUI_DataModel.csdl|res://*/Models.Bryan_GUI_DataModel.ssdl|res://*/Models.Bryan_GUI_DataModel.msl;provider=System.Data.SqlClient;provider connection string="data source=(LocalDB)\MSSQLLocalDB;attachdbfilename=|DataDirectory|\sspnet-Bryan_GUI-20180405013351.mdf;initial catalog=sspnet-Bryan_GUI-20180405013351;integrated security=true;providername=System.Data.SqlClient""/>
54 </connectionStrings>
```

The connection string is far too long to show but here is the fist third of the string. The string makes and establishes the connection to the database. The string is given to us by the host of the database, in our case Winnhost. The string is the reason we can connect stored procedures to a variable in out files.

One more thing I would like to highlight is a section of CSHTML code that creates a grid on checkboxes.

```

7   <h2>StudyArmDetailsView</h2>
8
9   <div>
10    <table class="table">
11      <tr>
12          <a href="#">@Html.ActionLink("View Appointment", "Appointments", new { studyArmPID = 2 }, new { @class = "btn btn-primary" })</a>
13          <th>Name</th>
14          <th>Duration</th>
15          <th>Visit #</th>
16      <tr>
17          <td>@foreach (var vis in Model.visits)
18          {
19              <th>Visit #@vis.visitID</th>
20          }
21      <tr>
22          <td>@foreach (var item in Model.activities)
23          {
24              <td>@item.actName</td>
25              <td>@item.actDuration</td>
26              <td>@foreach (var vis in Model.visits)
27              {
28                  <td><input class="form-check-input position-static" type="checkbox" id="blankCheckbox" value="option1" aria-label="..."/></td>
29              }
30          }
31      </tr>
32  </tbody>
33 </table>
34 </div>
35
36
37

```

Name	Duration	Visit #1	Visit #2	Visit #3	Visit #4	Visit #5	Visit #6	Visit #7	Visit #8	Visit #9	Visit #10
Blood Draw	60	<input type="checkbox"/>									
Random Transfusion	60	<input type="checkbox"/>									
Physical	60	<input type="checkbox"/>									
EKG	60	<input type="checkbox"/>									
Checkup	60	<input type="checkbox"/>									

Subject Information

This section of code took me a good amount of time to create. Its purpose is to make it easier for the user to create appointments by just checking the boxes. At this moment we are still unsure if we should use it. But regardless I think it should be seen if it does not make it to the final product.

5.2.3 Client-Side Programming

There are several executions that are being executed from the client side. Some examples can be seen in the add screens and other sections of the programs. The way it worked was that we sent a request to the back-end and received a response. Whether it worked the response we wanted, is a different story. That depends on how well we coded it.

5.3 Survey Questions

Outcome	Cesar Aleman	Bryan Sander
An ability to analyze a problem, identify, and define the computing requirements and specifications appropriate to its solution.	8	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	10
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.	9	8
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.	8	8