BSPC CLINIC DATABASE

COMP 5531 DATABASE PROJECT

GROUP LTC55311

Concordia University
8-18-2017

TABLE OF CONTENTS

Contents

SECTION 1.1 2
BSPC CLINIC Bahamas Sports Physio Center - INTRODUCTION 2
USER GUIDE 3
SECTION 1.2 14
ASSUMPTIONS 14
SECTION 1.3 17
ROLES AND RESPONSIBILITIES 17
SECTION 2.1 17
LIST OF TABLES 17
CREATE STATEMENTS 18
INPUT STATEMENTS 27
SECTION 2.2 28
DATABASE STRUCTURE 28
ER DIAGRAM 29
SECTION 2.3 29
TRIGGERS 29
STORED PROCEDURES 37
ROUTINES 52
SECTION 3.1 53
PHP SCRIPT 53
SECTION 3.2 54
REPORTS 54

SECTION 1.1

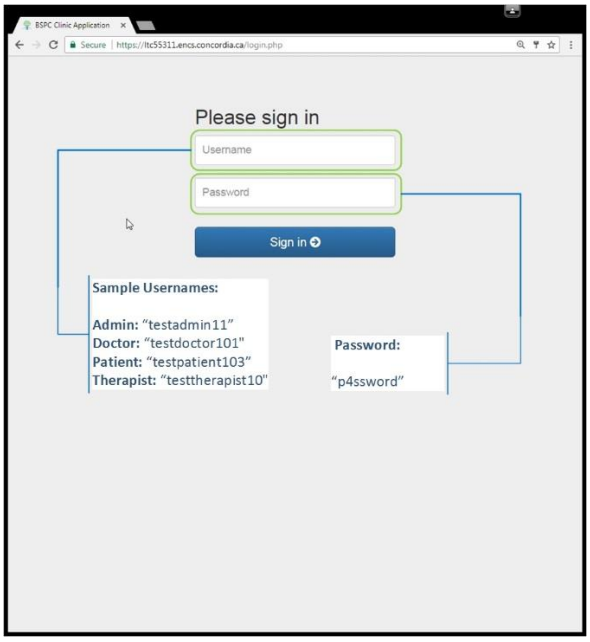
BSPC CLINIC Bahamas Sports Physio Center - INTRODUCTION

Bahamas Sports Physio Center – BSPC caters to the physio treatment needs for patients with varying physio problems. Patients are referred to the clinic by an external specialist who also recommends a course of treatment. The physio center handles receiving the patient, registering them, scheduling therapy appointments, administering physio care and keeping track of patient records.

The physio center employs receptionist/admin personnel to manage day to day activities and doctors/ therapists for administering care. Our mandate was to develop and implement a database system to store all the physio center information as well as an online application to interface with the underlying database. Below is a description of the application and its usage as well as implementation details for the database.

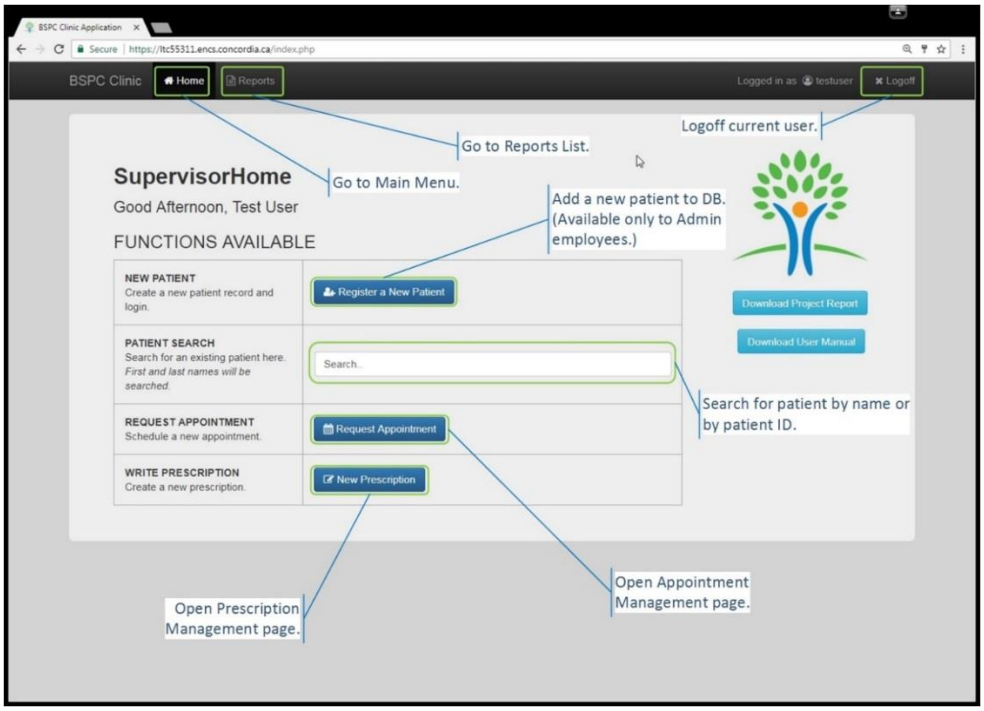
USER GUIDE

1. Login on the system – provide user credentials



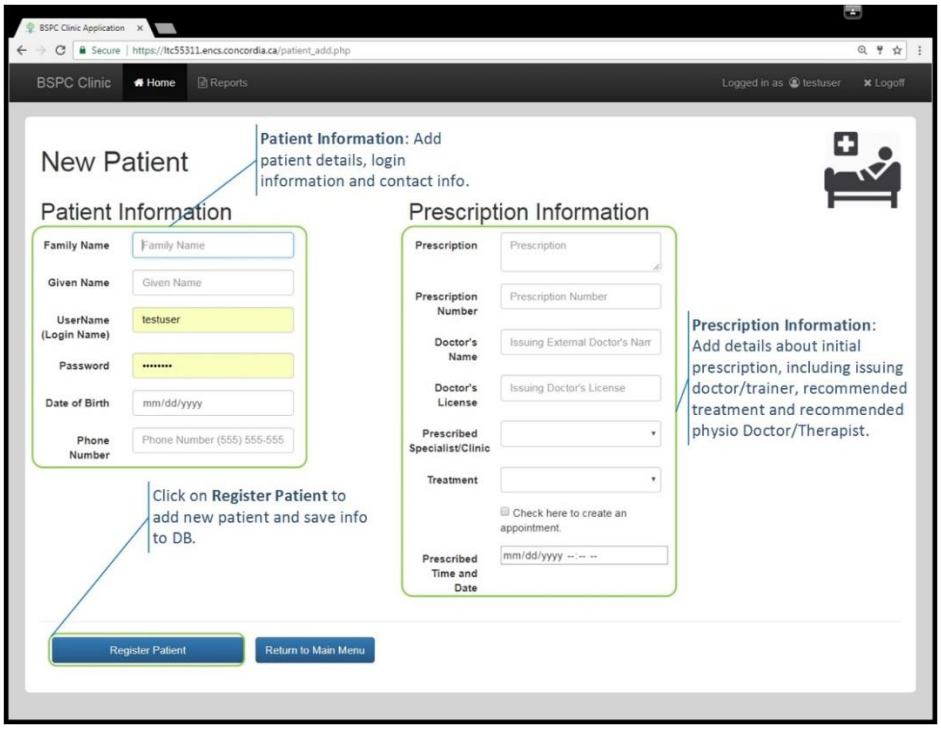
2. Landing page (index page)

For different users, there are different levels of authority specified for them in the page. Following is a view of the index page with all features displayed and descriptions provided.

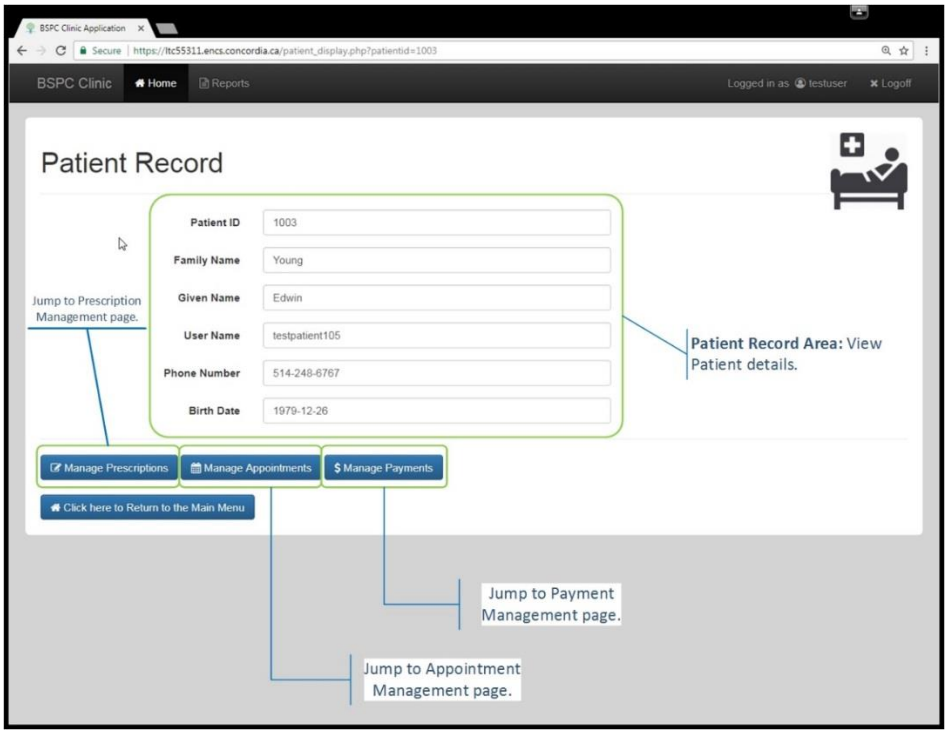


Roles for Receptionist:

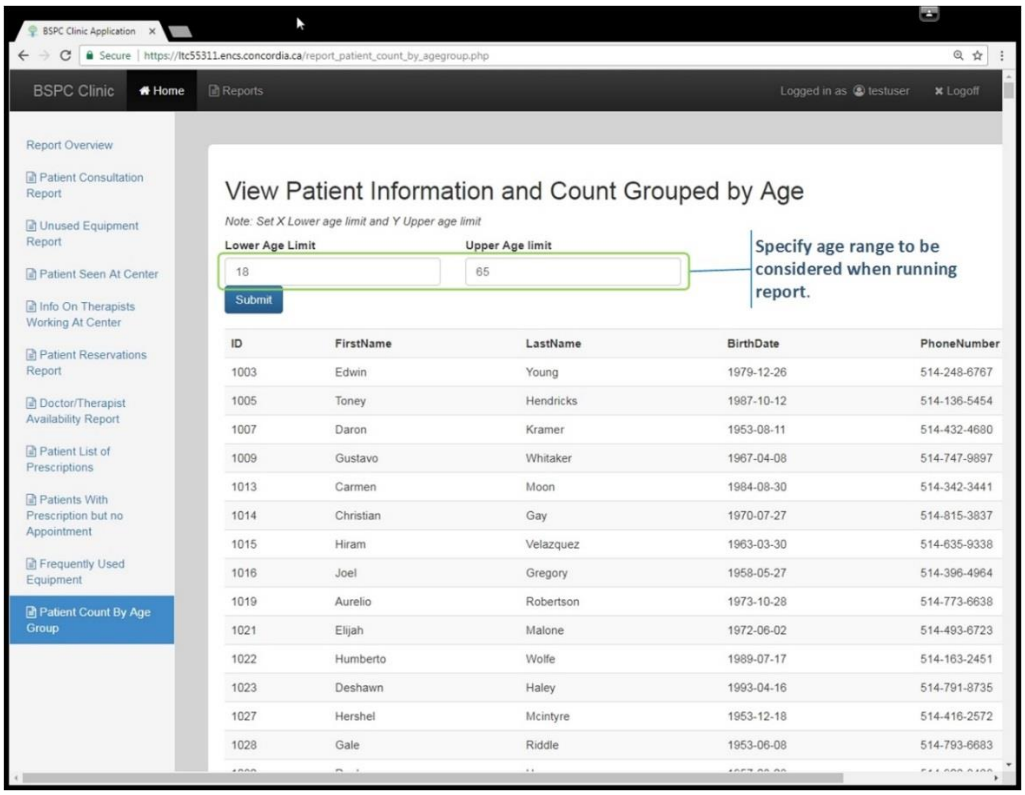
- **Register a new patient**



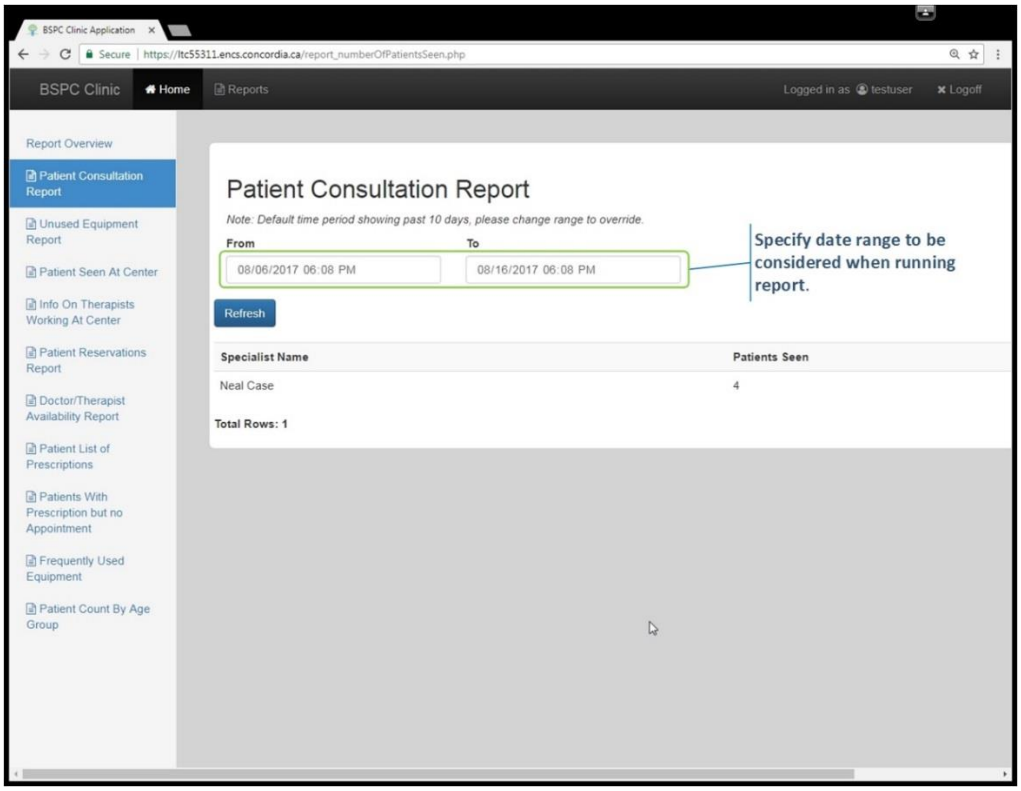
- **Search for a patient**



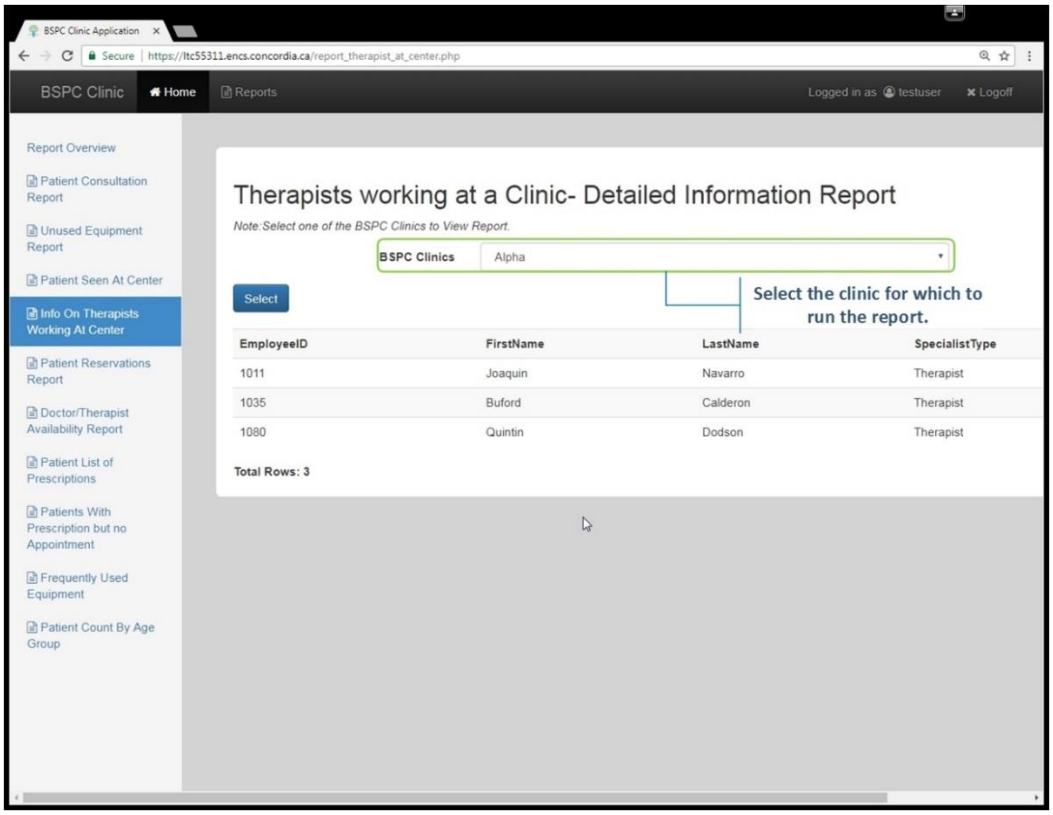
- **Download Reports**
  - **Report: Patient Count by Age group**



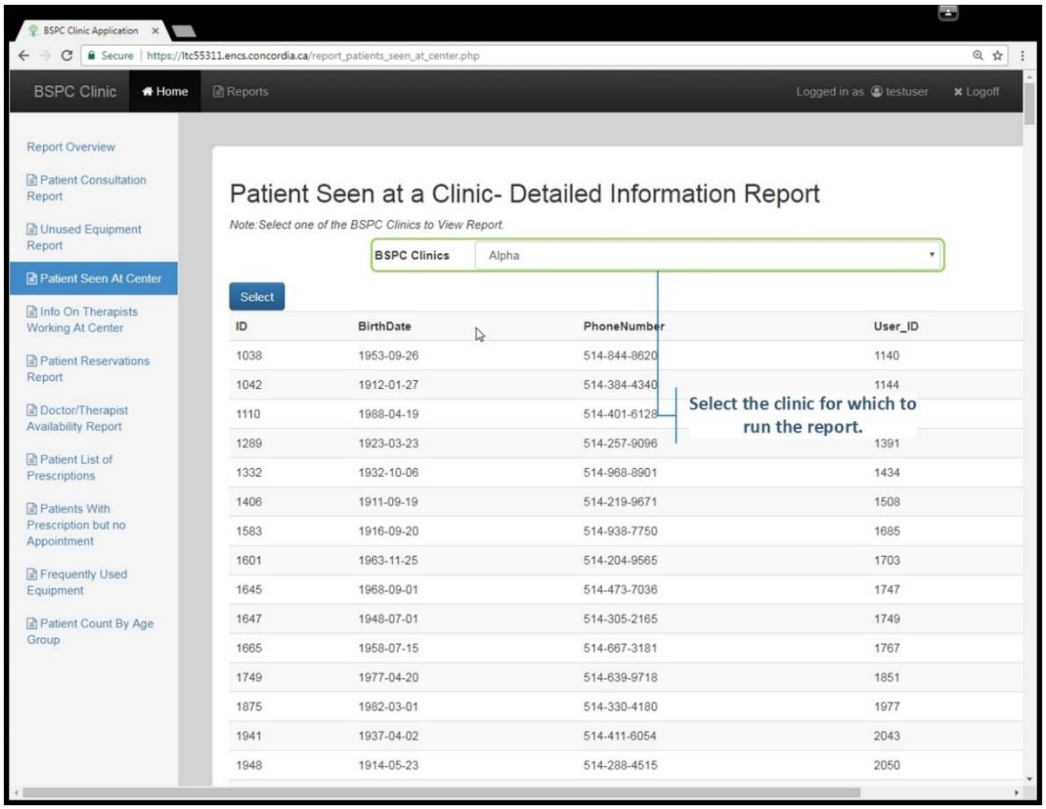
- **Report: Number of Patients seen**



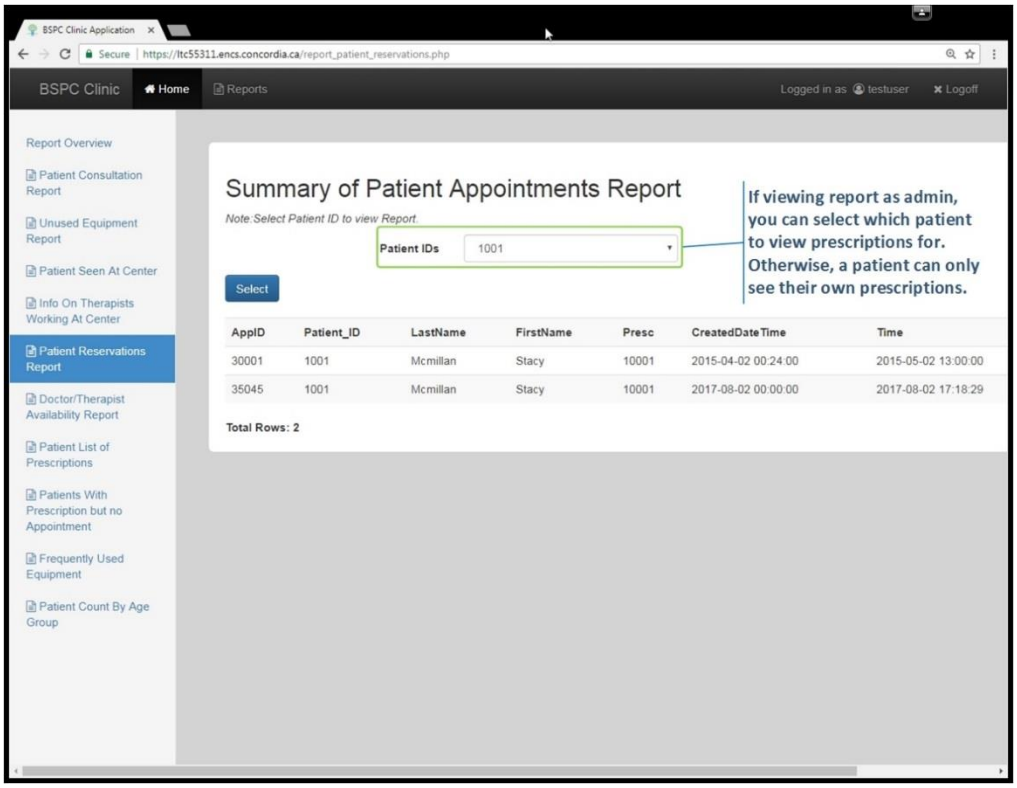
- **Report: Therapists at Center**



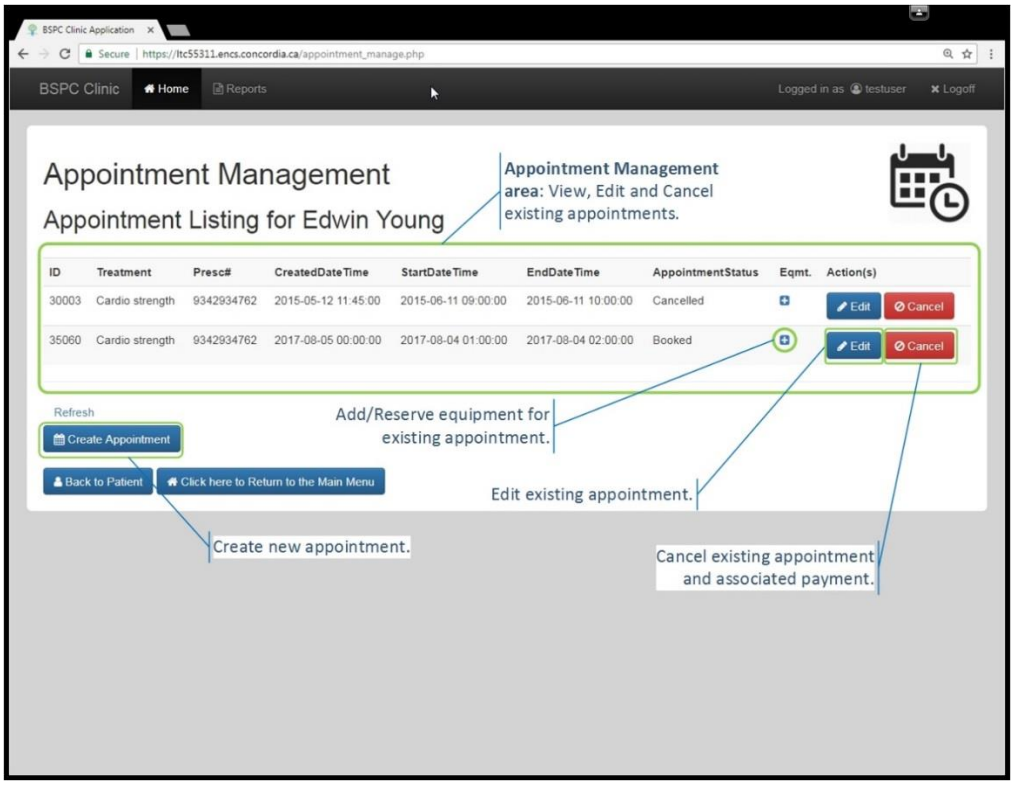
- **Report: Patients seen at Center**



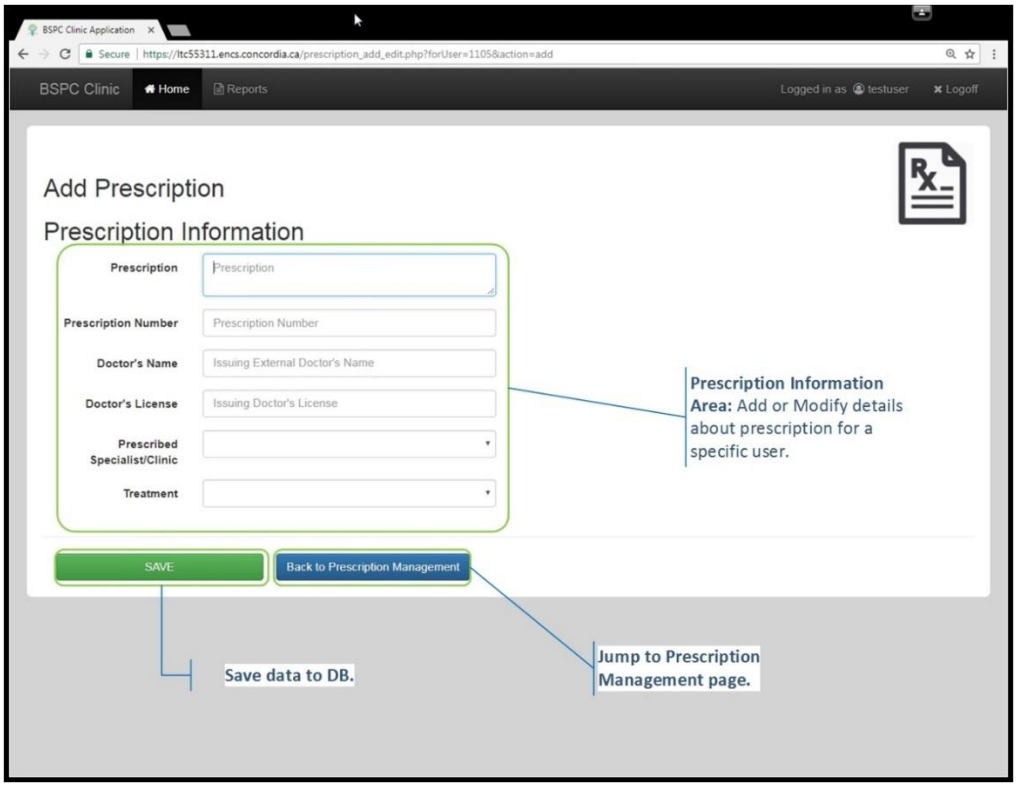
- **Report: Patients Reservations**



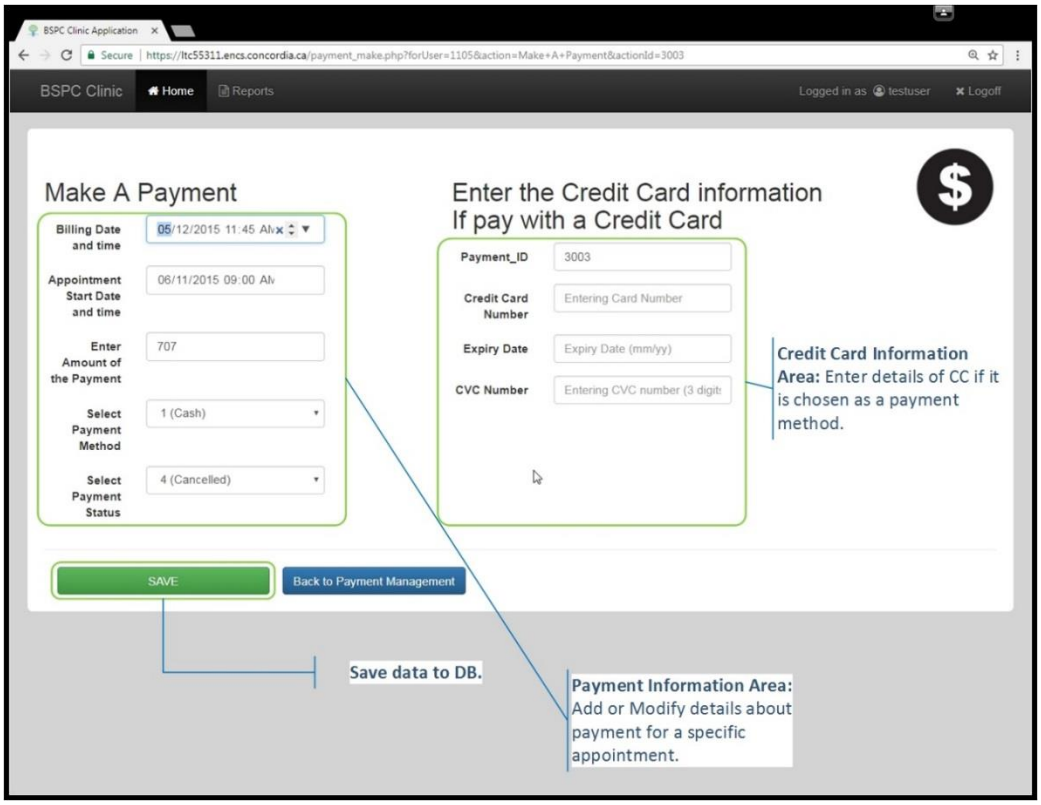
- **Manage an Appointment for a Patient**



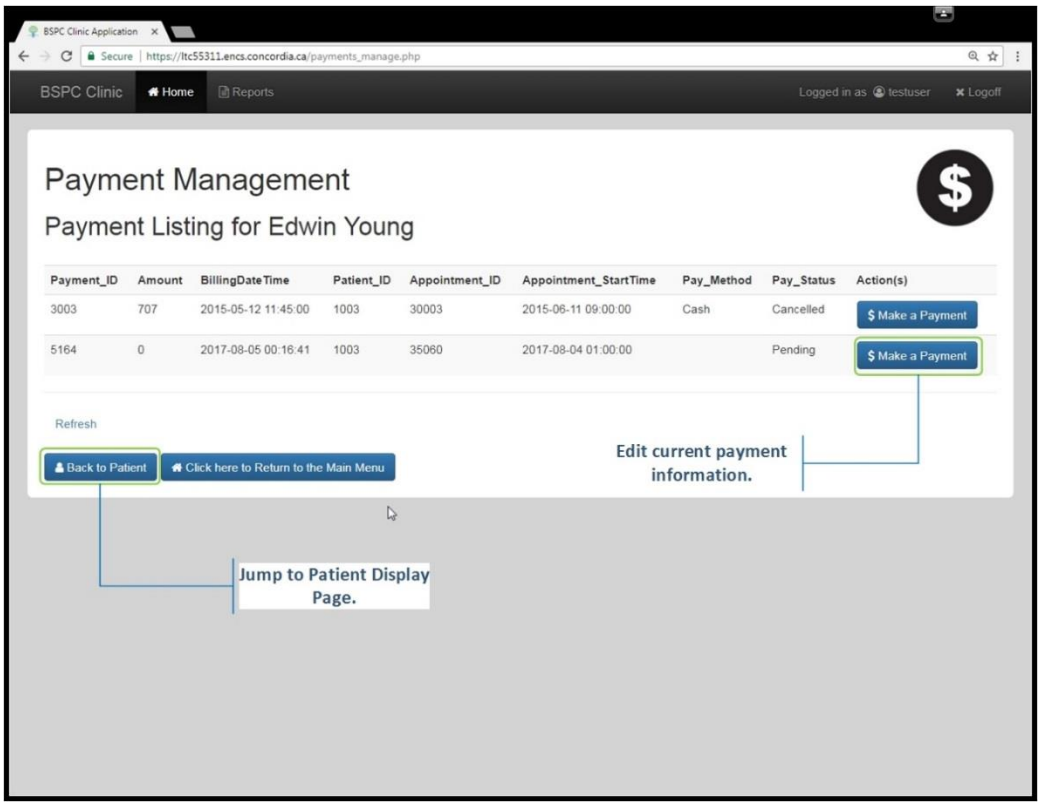
- **Add a Prescription for a Patient**



- **Process a Payment for a Patient**

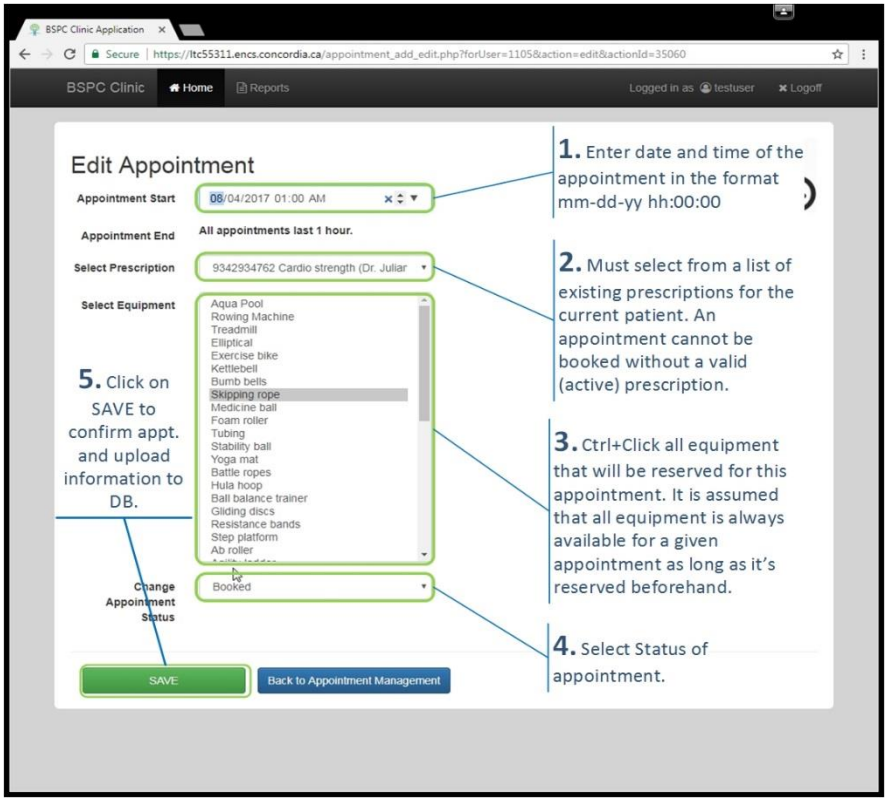


- **Manage a Payment for a Patient**

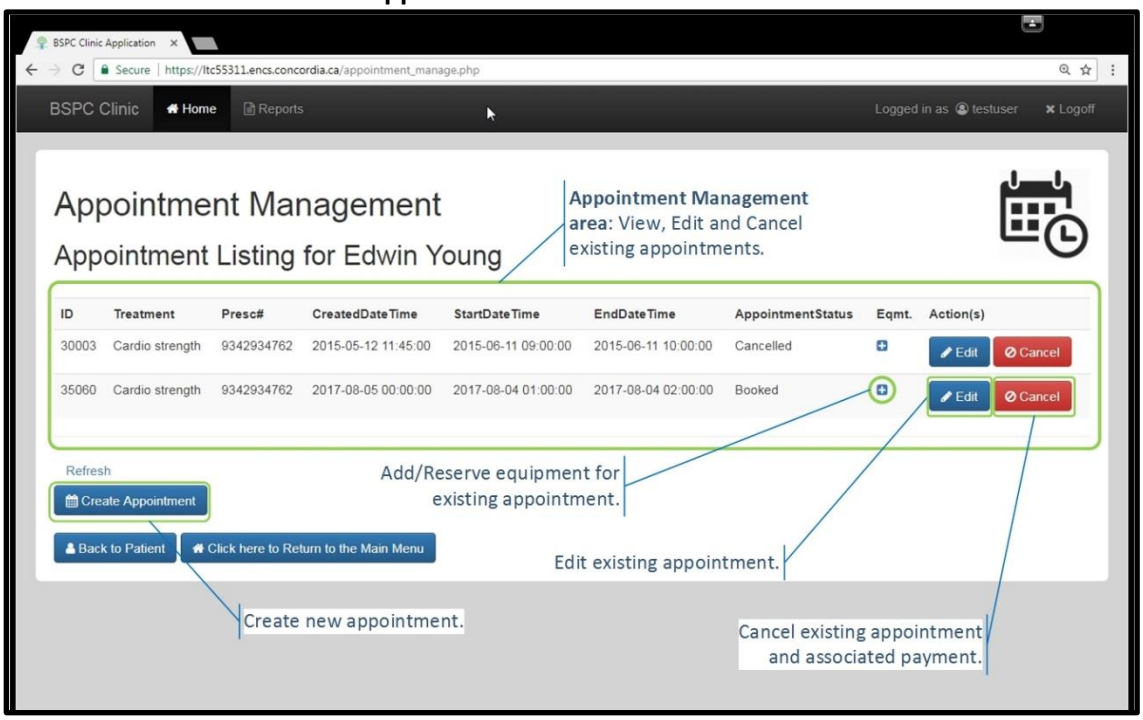


Roles for Patient:

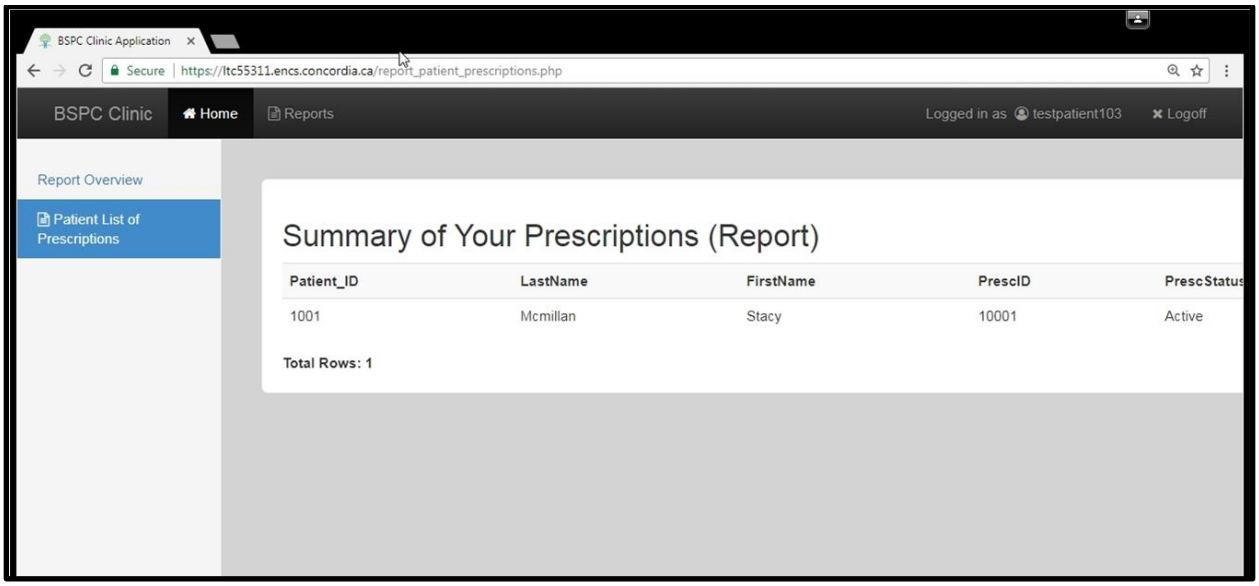
- **Request an Appointment**



- **Cancel an Existed Appointment**



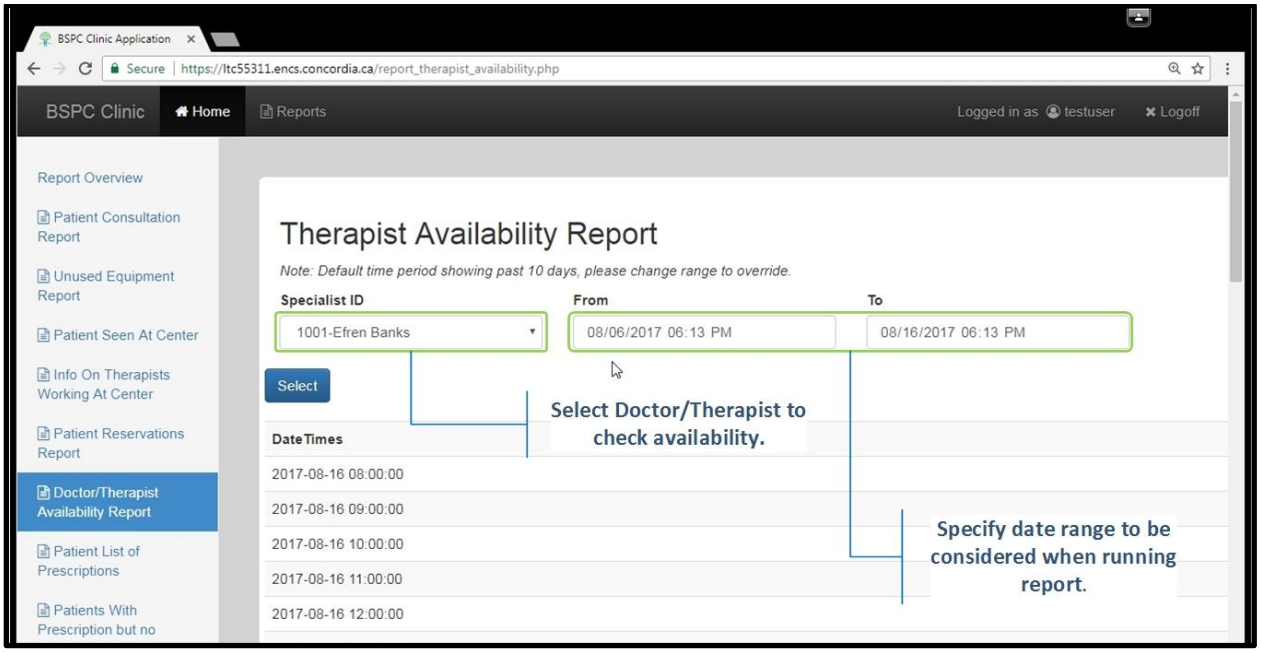
- **Download Prescription Report**



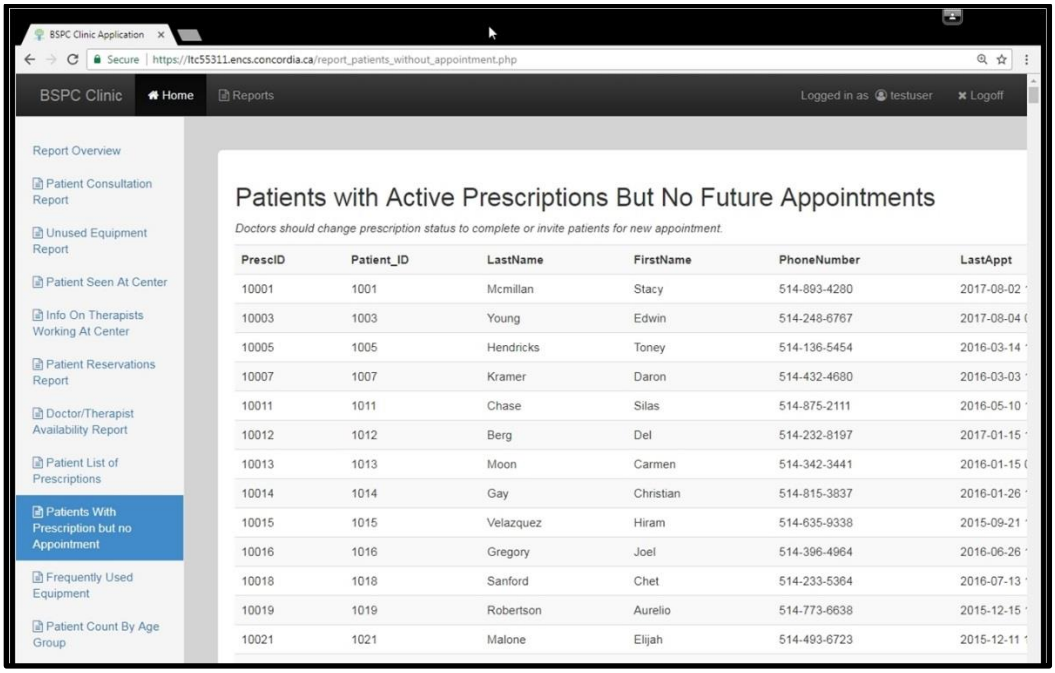
- **View prescription history report (See above for procedure)**

Roles for Doctors:

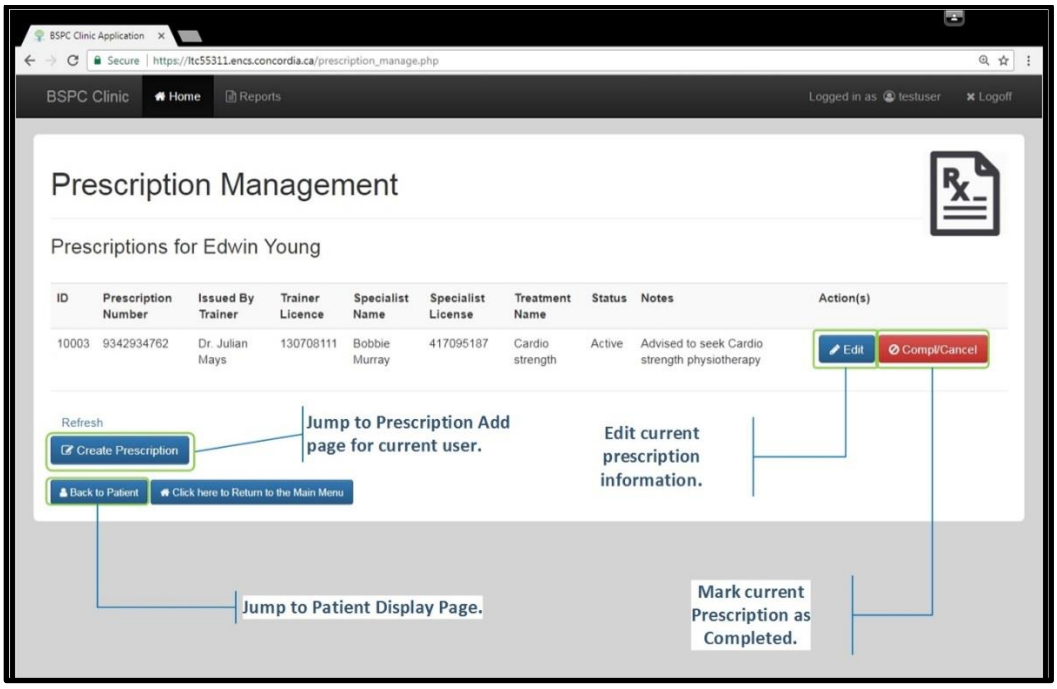
- **Search for a patient (See above for procedure)**
- **Download Reports**
  - Report: Unused equipment (See above)
  - Report: Patient reservation (See above)
  - Reports: Doc/therapist availability



- **Report: Patient with prescription w/o appointment**



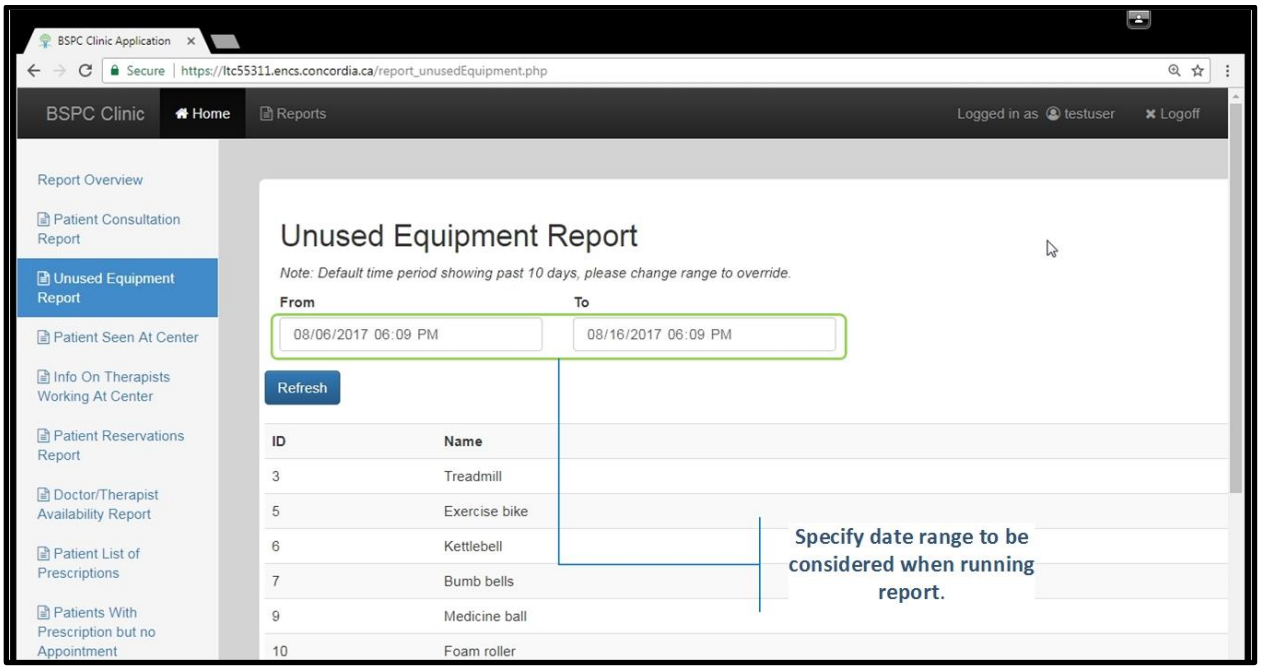
- **Manage a Prescription for a Patient**



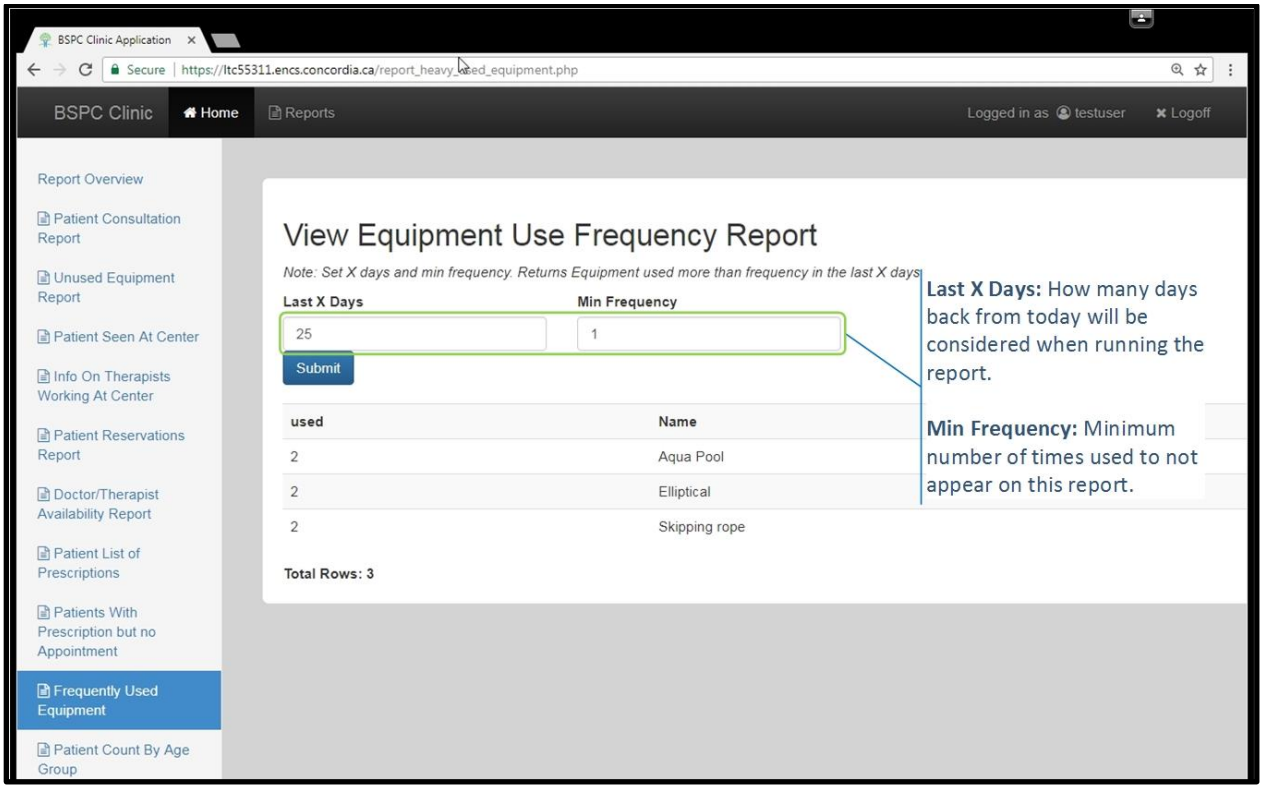
Roles for Therapists:

- **Search for a patient (See above for procedure)**
- **Download Reports**

  - Report: Unused Equipment



- Report: Patient reservation (See above)
- Report: Doc/therapist availability (See above)
- Report: Frequently used equipment



- **Manage a Prescription for a Patient (See above for procedure)**

SECTION 1.2

ASSUMPTIONS

Notes:
“+” -> assumptions handled by SQL procedure call.
“*” -> assumptions handled by PHP.
There are 23 tables discussed as below.

General Assumptions:

'Appointment Table':

+ A patient (or a receptionist on behalf of the patient) can attempt to create an appointment at any time

+ To make an appointment, an active (prescription_status = 1) prescription_id is required

+ An appointment can only be made up to eight weeks in advance

+ A bill is created in the 'Payment' table on the date the appointment is created

+ All appointments start off with the default status '1' (Booked)

+ When an appointment is cancelled, Appointment.Appointment_Status must be set to '3' (Cancelled) and the associated payment in the "Payment" table must have Payment.Payment_Status set to '4' (Cancelled)

+ If an appointment is honoured or there is a no-show (missed), the associated payment must be made

+ An appointment status can be changed to '3' (Cancelled) until the day of the appointment. Otherwise, the status can only be changed to either '2' (Attended) or '4' (Missed).

+ Specialists cannot have overlapping appointments

+ All appointments last one hour, between the hours of 08:00 and 17:00, and begin on the hour.

+ A new appointment cannot be created if the patient has a previous rejected payment.

'Appointment_Equipment Table':

+ All facilities are assumed to have as many pieces of equipment as needed, eliminating any reservation conflicts.

* An appointment can have multiple pieces of equipment associated with it.

'Appointment_Treatment Table'

* An appointment can have multiple treatments associated with it

+ If the associated prescription_id for an appointment has a treatment specified, only that treatment can be reserved for the appointment. Otherwise, any number of treatments can be selected.

'Credit_Card Table':

* When sending information for verification, information is extracted from this table

'Employee Table':

+ Each employee has an associated center_id

+ If an employee has Employee_Type_ID = 2 (Specialist), then they have to be in the 'Specialist' table

+ All employees have a user ID Employee.User_ID is equal to their User.id Users don't share the same access rights

'Patient Table' :

+ A patient can only be added if they are over 18 years old

* A patient can only make or cancel appointments, and choose equipment/treatment for appointments

* A patient can only be added if they present themselves with a prescription (this initial prescription is recorded in Patient.Prescription_ID)

+ Initial prescriptions (the ones required to register at the center in the first place) are made by external trainers/health-care specialists that are assumed not be employed by the center. As such, we keep a record of their provided name and license number, but these are not foreign keys to another table in our system.

'Payment Table' :

+ Once CC verified, the associated payment is changed from 2 (Pending) to either 1 (Paid) or 3 (Rejected)// handled by an event in SQL before update confirm status != paid

* If Payment.Payment_Method = 4 (Credit Card), then the CC information is saved in Credit_Card table

+ If an appointment is honoured or there is a no-show, the associated payment must be made. (You can't cancel a payment after the date of the appointment)

+ Cheque, Cash and Debit payments are assumed to all clear immediately and don't require verification (Payment.Payment_Status_ID = 1-paid)

'Prescription Table':

+ Prescriptions can have only one treatment

+ Initial prescriptions (the ones required to register to the center in the first place) are made by external trainers/health-care specialists that are assumed not be employed by the center. As such, we keep a record of their provided name and license number, but these are not foreign keys to another table in our system. (Same assumption as patient table)

+ Specialist_ID is a foreign key to the Specialist table

+ Treatment_ID and Specialist_ID can be Null allowing the patient to book an appointment with any specialist and for any treatment.

+ Prescription status must be checked before an appointment can be booked. If a prescription_status is 1(Active) the patient can book appointments. If the status is 2(Completed) no appointments can be booked on the prescription.

'Specialist Table' :

+ There are two types of specialists distinguished by 'SpecialistType_ID' of 1 - Doctors or 2-Therapists.

+ The SpecialistType_ID is determined by the years of experience the specialist has.

+ The Employee_ID is a foreign key and primary key to the specialist table.

+ The Employee_ID is used to retrieve the specialist's name from Employee table for booking appointments.

'User Table':

+ Type_ID specifies the access/and view of the user.

+ User.ID is a foreign key in the Employee and Patient tables for existing accounts.

SECTION 1.3

ROLES AND RESPONSIBILITIES

Application Development- **Team Lead: Brendan Wood**
Contributors: Max Carrénard, Aiken Chung, Wambui Kinyanjui

Database Design- **Team Lead: Wambui Kinyanjui**
Contributors: Max Carrénard, Aiken Chung, Brendan Wood

Data Generation- **Team Lead: Max Carrénard**
Contributors: Wambui Kinyanjui, Aiken Chung, Brendan Wood

Database Implementation- **Team Lead: Aiken Chung**
Contributors: Wambui Kinyanjui, Brendan Wood, Max Carrénard

SECTION 2.1

LIST OF TABLES

Appointment:	Record of Appointment date, times and status for patients. Links to equipment, treatment and payment tables made for each patient and status.
Appointment-Equipment:	Record of equipment required for each appointment
Appointment_Status:	Record of appointment statuses
Appointment_Times:	Helper table: Record of Appointment times between 9-5pm. Used for therapist and doctor availability
Appointment_Treatment:	Record of treatment required for each appointment
CCVerification:	Verification table, populated every day with credit card details used for the day's transactions
Center:	Record of the four BSPC clinics, shows address and contact information.
CreditCard:	Record of credit card information supplied at time of credit payments
DateTimeBag:	Helper table, dynamically populated with all possible appointment times for the search period provided by user. Used for therapist and doctor availability
Employee:	Record of clinic employees: doctors, therapist, receptionists, etc.
EmployeeType:	Employee types differentiating specialists from receptionists and other
Equipment:	Record of all equipment available at the clinic
Patient:	Record of all patients who have been treated at the clinic
Payment:	Record of all transactions, status and methods of payment.
PaymentMethod:	Record of all available payment methods
PaymentStatus:	Record of payment status: paid, pending, rejected or cancelled
Prescription:	Record of all prescriptions, links to patient and specialist
PrescriptionStatus:	Prescription status: active/ongoing or completed for past treatments
Specialist:	Record of all specialist ID, license numbers , years of experience

SpecialistType:	Differentiates between doctors and therapists
Treatment:	Record of treatment options available at the clinic
User:	Record of all users with access to the BSPC application, displays user credentials, names and login information
UserType:	Differentiates access types available to users

CREATE STATEMENTS

#Appointment

```
CREATE TABLE `Appointment` (
  `ID` INT(11) NOT NULL AUTO_INCREMENT,
  `Patient_ID` INT(11) NOT NULL,
  `Prescription_ID` INT(11) NOT NULL,
  `CreatedDateTime` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `StartDateTime` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
  `EndDateTime` DATETIME NOT NULL,
  `Status_ID` INT(11) NOT NULL DEFAULT '1',
  PRIMARY KEY (`ID`),
  KEY `fk__idx` (`Prescription_ID`),
  KEY `fk_Appointment_Status_ID_idx` (`Status_ID`),
  KEY `fk_Patient_ID_idx` (`Patient_ID`),
  CONSTRAINT `fk_Appointment_Status_ID` FOREIGN KEY (`Status_ID`)
REFERENCES `Appointment_Status` (`ID`) ON DELETE NO ACTION ON UPDATE
NO ACTION,
  CONSTRAINT `fk_Patient_ID` FOREIGN KEY (`Patient_ID`) REFERENCES
`Patient` (`ID`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `fk_Prescription_ID` FOREIGN KEY (`Prescription_ID`)
REFERENCES `Prescription` (`ID`) ON DELETE NO ACTION ON UPDATE NO
ACTION
) ENGINE=InnoDB AUTO_INCREMENT=35058 DEFAULT CHARSET=latin1;
```

#Appointment-Equipment

```
CREATE TABLE `Appointment_Equipment` (
  `ID` INT(11) NOT NULL AUTO_INCREMENT,
  `Appointment_ID` INT(11) NOT NULL,
  `Equipment_ID` INT(11) NOT NULL,
  PRIMARY KEY (`ID`),
  KEY `fk_Appointment_Equipment_Equipment1_idx` (`Equipment_ID`),
  KEY `fk_Appointment_Equipment_Appointment1_idx` (`Appointment_ID`),
  CONSTRAINT `fk_Appointment_Equipment_Appointment1` FOREIGN KEY
(`Appointment_ID`) REFERENCES `Appointment` (`ID`) ON DELETE NO ACTION
ON UPDATE NO ACTION,
  CONSTRAINT `fk_Appointment_Equipment_Equipment1` FOREIGN KEY
(`Equipment_ID`) REFERENCES `Equipment` (`ID`) ON DELETE NO ACTION ON
UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=9014 DEFAULT CHARSET=latin1;
```

#Appointment_Status

```
CREATE TABLE `Appointment_Status` (
  `ID` INT(11) NOT NULL AUTO_INCREMENT,
  `Name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=latin1;
```

#Appointment_Times

```
CREATE TABLE `Appointment_Times` (
  `ID` INT(11) NOT NULL AUTO_INCREMENT,
  `Times` time NOT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=latin1;
```

#Appointment_Treatment

```
CREATE TABLE `Appointment_Treatment` (
  `ID` INT(11) NOT NULL AUTO_INCREMENT,
  `Appointment_ID` INT(11) NOT NULL,
  `Treatment_ID` INT(11) NOT NULL,
  PRIMARY KEY (`ID`),
  KEY `fk_Appointment_ID_idx` (`Appointment_ID`),
  KEY `fk_Treatment_ID_idx` (`Treatment_ID`),
  CONSTRAINT `fk_Appointment_ID` FOREIGN KEY (`Appointment_ID`)
REFERENCES `Appointment` (`ID`) ON DELETE NO ACTION ON UPDATE NO
ACTION,
  CONSTRAINT `fk_Treatment_ID` FOREIGN KEY (`Treatment_ID`) REFERENCES
`Treatment` (`ID`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=3004 DEFAULT CHARSET=latin1;
```

#CCVerification

```
CREATE TABLE `CCVerification` (
  `ID` INT(11) NOT NULL,
  `PaymentDATETIME` DATETIME NOT NULL,
  `VerificationDATETIME` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `Number` VARCHAR(19) NOT NULL,
  `ExpiryDate` VARCHAR(6) NOT NULL,
  `CVC` INT(5) NOT NULL,
  `Amount` double DEFAULT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

#Center

```
CREATE TABLE `Center` (
  `ID` INT(11) NOT NULL,
```

```
  `Name` VARCHAR(45) DEFAULT NULL,
  `Address` VARCHAR(45) DEFAULT NULL,
  `PhoneNumber` VARCHAR(45) DEFAULT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

#CreditCard

```
CREATE TABLE `CreditCard` (
  `Payment_ID` INT(11) NOT NULL,
  `Number` VARCHAR(19) NOT NULL,
  `ExpiryDate` VARCHAR(6) NOT NULL,
  `CVC` INT(5) NOT NULL,
  PRIMARY KEY (`Payment_ID`),
  CONSTRAINT `Payment_ID` FOREIGN KEY (`Payment_ID`) REFERENCES
`Payment`
  (`ID`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

#DateTimeBag

```
CREATE TABLE `DateTimeBag` (
  `ID` INT(11) NOT NULL AUTO_INCREMENT,
  `DATETIMES` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

#Employee

```
CREATE TABLE `Employee` (
  `ID` INT(11) NOT NULL AUTO_INCREMENT,
  `Center_ID` INT(11) DEFAULT NULL,
```

```
  `EmployeeType_ID` INT(11) DEFAULT NULL,
  `User_ID` INT(11) DEFAULT NULL,
  PRIMARY KEY (`ID`),
  KEY `fk_Center_ID_idx` (`Center_ID`),
  KEY `fk_Employee_Type_ID_idx` (`EmployeeType_ID`),
  KEY `fk_User_ID_idx` (`User_ID`),
  CONSTRAINT `fk_Center_ID` FOREIGN KEY (`Center_ID`) REFERENCES
`Center` (`ID`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `fk_Employee_Type_ID` FOREIGN KEY (`EmployeeType_ID`)
REFERENCES `EmployeeType` (`ID`) ON DELETE NO ACTION ON UPDATE NO
ACTION,
  CONSTRAINT `fk_User_ID` FOREIGN KEY (`User_ID`) REFERENCES `User`
(`ID`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=1101 DEFAULT CHARSET=latin1;
```

#EmployeeType

```
CREATE TABLE `EmployeeType` (
  `ID` INT(11) NOT NULL,
  `Name` VARCHAR(45) DEFAULT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

#Equipment

```
CREATE TABLE `Equipment` (
  `ID` INT(11) NOT NULL,
  `Name` VARCHAR(45) DEFAULT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

#Patient

```
CREATE TABLE `Patient` (
  `ID` INT(11) NOT NULL AUTO_INCREMENT,
  `BirthDate` date NOT NULL,
  `PhoneNumber` VARCHAR(15) DEFAULT NULL,
  `InitialPrescription_ID` INT(11) NOT NULL,
  `User_ID` INT(11) DEFAULT NULL,
  PRIMARY KEY (`ID`),
  KEY `fk_Prescription_ID_idx` (`InitialPrescription_ID`),
  KEY `fk_User_ID_idx` (`User_ID`),
  CONSTRAINT `Prescription_ID` FOREIGN KEY (`InitialPrescription_ID`)
  REFERENCES `Prescription` (`ID`) ON DELETE NO ACTION ON UPDATE NO
ACTION,
  CONSTRAINT `User_ID` FOREIGN KEY (`User_ID`) REFERENCES `User`
(`ID`)
  ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=3259 DEFAULT CHARSET=latin1;
```

#Payment

```
CREATE TABLE `Payment` (
  `ID` INT(11) NOT NULL AUTO_INCREMENT,
  `Amount` double DEFAULT NULL,
  `BillingDATETIME` DATETIME DEFAULT NULL,
  `PaymentMethod_ID` INT(11) DEFAULT NULL,
  `PaymentStatus_ID` INT(11) NOT NULL DEFAULT '2',
  `Appointment_ID` INT(11) NOT NULL,
  PRIMARY KEY (`ID`),
  KEY `fk_Payment_Method_ID_idx` (`PaymentMethod_ID`),
  KEY `fk_Payment_Status_ID_idx` (`PaymentStatus_ID`),
  KEY `fk_Appointment_ID_idx` (`Appointment_ID`),
  CONSTRAINT `fk_Payment_Status_ID` FOREIGN KEY (`PaymentStatus_ID`)
```

```
  REFERENCES `PaymentStatus` (`ID`) ON DELETE NO ACTION ON UPDATE NO
ACTION,
  CONSTRAINT `fk_Payment_Method_ID` FOREIGN KEY (`PaymentMethod_ID`)
  REFERENCES `PaymentMethod` (`ID`) ON DELETE NO ACTION ON UPDATE NO
ACTION
) ENGINE=InnoDB AUTO_INCREMENT=5162 DEFAULT CHARSET=latin1;
```

#PaymentMethod

```
CREATE TABLE `PaymentMethod` (
  `ID` INT(11) NOT NULL,
  `Name` VARCHAR(45) DEFAULT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

#PaymentStatus

```
CREATE TABLE `PaymentStatus` (
  `ID` INT(11) NOT NULL,
  `Name` VARCHAR(45) DEFAULT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

#Prescription

```
CREATE TABLE `Prescription` (
  `ID` INT(11) NOT NULL AUTO_INCREMENT,
  `PrescriptionNumber` double NOT NULL,
  `Patient_ID` VARCHAR(45) DEFAULT NULL,
  `IssuedByTrainer` VARCHAR(45) DEFAULT NULL,
  `IssuedByLicence` VARCHAR(45) DEFAULT NULL,
  `Specialist_ID` INT(11) DEFAULT NULL,
  `Treatment_ID` INT(11) DEFAULT NULL,
```

```
  `PrescriptionStatus_ID` INT(11) NOT NULL,
  `Notes` VARCHAR(500) DEFAULT NULL,
  PRIMARY KEY (`ID`),
  KEY `fk_Specialist_ID_idx` (`Specialist_ID`),
  KEY `fk_Treatment_ID_idx` (`Treatment_ID`),
  KEY `fk_Prescription_Status_ID_idx` (`PrescriptionStatus_ID`),
  CONSTRAINT `fk_Prescription_Status_ID` FOREIGN KEY
(`PrescriptionStatus_ID`)
  REFERENCES `PrescriptionStatus` (`ID`) ON DELETE NO ACTION ON UPDATE
NO ACTION,
  CONSTRAINT `Specialist_ID` FOREIGN KEY (`Specialist_ID`) REFERENCES
`Specialist`
  (`Employee_ID`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `Treatment_ID` FOREIGN KEY (`Treatment_ID`) REFERENCES
`Treatment`
  (`ID`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=12029 DEFAULT CHARSET=latin1;
```

#PrescriptionStatus

```
CREATE TABLE `PrescriptionStatus` (
  `ID` INT(11) NOT NULL,
  `Description` VARCHAR(45) DEFAULT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

#Specialist

```
CREATE TABLE `Specialist` (
  `Employee_ID` INT(11) NOT NULL,
  `LicenceNumber` INT(11) NOT NULL,
  `SpecialistType_ID` INT(11) NOT NULL,
  `YearsExperience` VARCHAR(45) NOT NULL,
```

```
  PRIMARY KEY (`Employee_ID`),
  KEY `fk_Specialist_Type_ID_idx` (`SpecialistType_ID`),
  CONSTRAINT `fk_Specialist_Type_ID` FOREIGN KEY (`SpecialistType_ID`)
  REFERENCES `SpecialistType` (`ID`) ON DELETE NO ACTION ON UPDATE NO
ACTION,
  CONSTRAINT `fk_Employee_ID` FOREIGN KEY (`Employee_ID`) REFERENCES
`Employee` (`ID`)
  ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

#SpecialistType

```
CREATE TABLE `SpecialistType` (
  `ID` INT(11) NOT NULL,
  `Name` VARCHAR(45) DEFAULT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

#TestTable

```
CREATE TABLE `TestTable` (
  `ID` INT(11) NOT NULL,
  `TestVal` VARCHAR(45) DEFAULT NULL,
  `Num` INT(11) DEFAULT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

#Treatment

```
CREATE TABLE `Treatment` (
  `ID` INT(11) NOT NULL,
  `Name` VARCHAR(45) DEFAULT NULL,
  PRIMARY KEY (`ID`)) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

#User

```
CREATE TABLE `User` (
  `ID` INT(11) NOT NULL AUTO_INCREMENT,
  `LoginID` VARCHAR(45) DEFAULT NULL,
  `FirstName` VARCHAR(45) DEFAULT NULL,
  `LastName` VARCHAR(45) DEFAULT NULL,
  `Password` VARCHAR(200) DEFAULT NULL,
  `Type_ID` VARCHAR(45) DEFAULT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB AUTO_INCREMENT=3175 DEFAULT CHARSET=latin1;
```

#UserType

```
CREATE TABLE `UserType` (
  `ID` INT(11) NOT NULL,
  `Description` VARCHAR(45) DEFAULT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

INPUT STATEMENTS

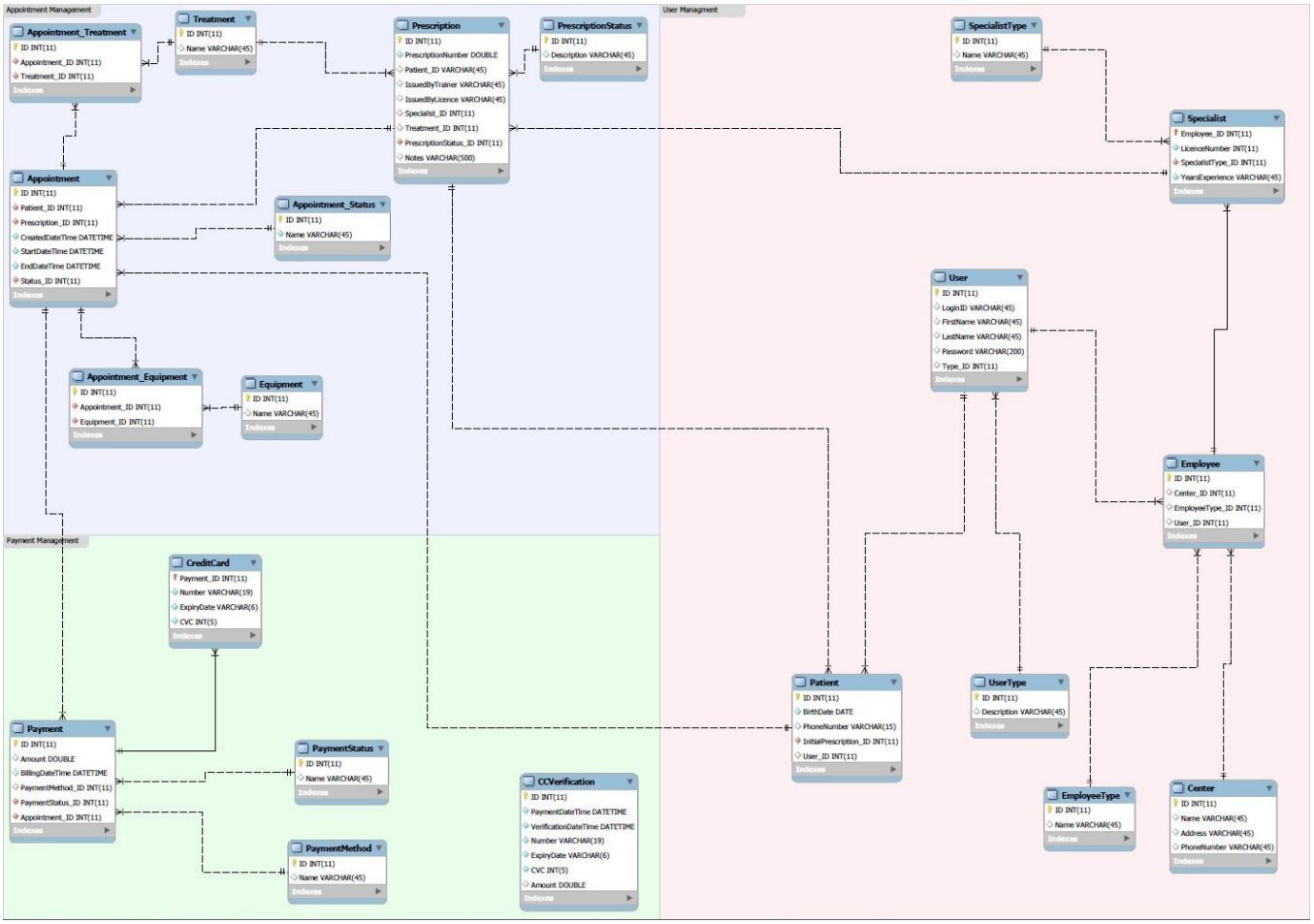
SEE SECONDARY REPORT ATTACHED

LINK TO LOCATION ON SITE

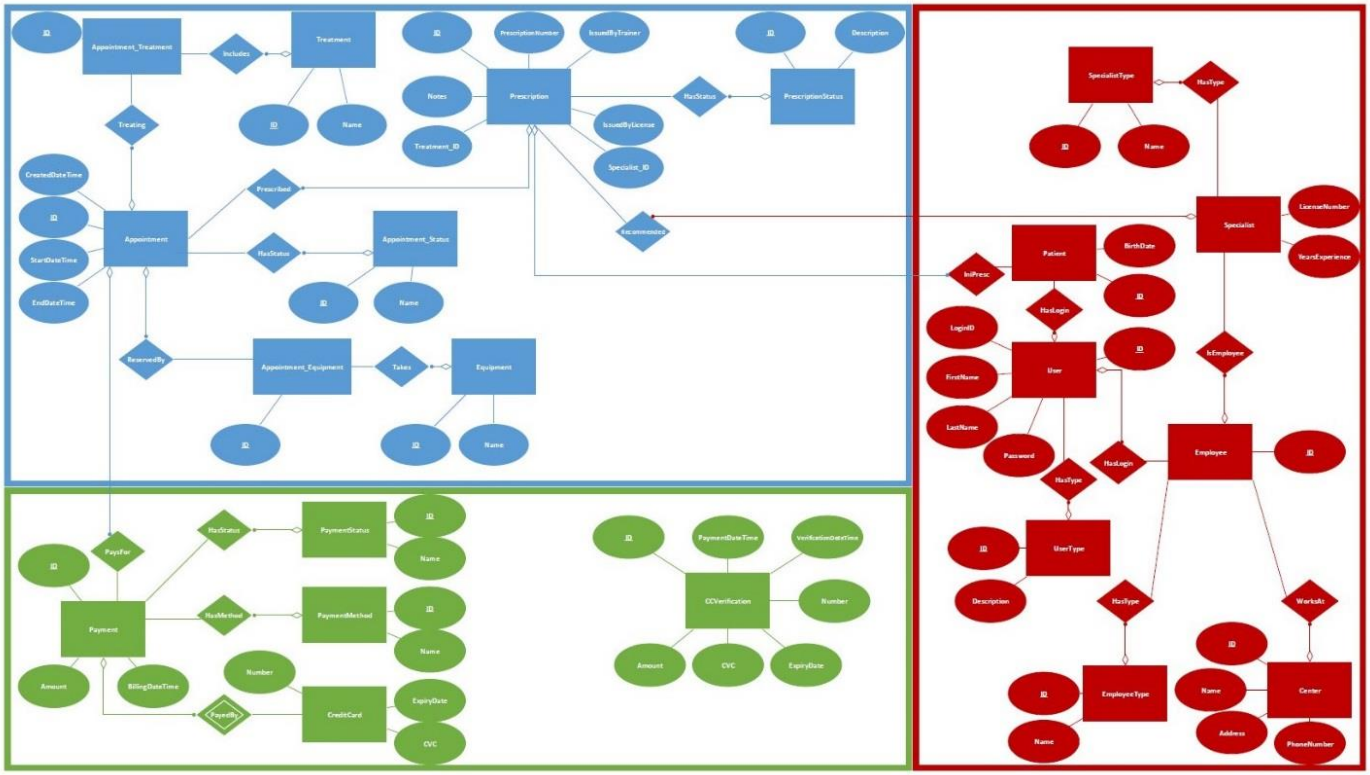
https://ltc55311.encs.concordia.ca/assets/documents/insert_statements_ltc55311_summer2017.pdf

SECTION 2.2

DATABASE STRUCTURE



ER DIAGRAM



SECTION 2.3

TRIGGERS

Appointment_Treatment_BEFORE_INSERT

```
CREATE DEFINER=`ltc55311`@`132.205.%.%` TRIGGER
`ltc55311`.`Appointment_Treatment_BEFORE_INSERT`
BEFORE INSERT ON `Appointment_Treatment`
FOR EACH ROW
BEGIN
     DECLARE definedTremantID int(11);
    SELECT Prescription.Treatment_ID
    From ltc55311.Prescription inner join ltc55311.Appointment
    ON Appointment.Prescription_ID = Prescription.ID
```

```
    WHERE NEW.Appointment_ID = Appointment.ID
    INTO definedTremantID;
    IF (definedTremantID!=NEW.Treatment_ID)
    THEN
           SET NEW.Treatment_ID = definedTremantID;
     END IF;
END

#Patient_BEFORE_INSERT
CREATE DEFINER=`ltc55311`@`132.205.%.%` TRIGGER
`ltc55311`.`Patient_BEFORE_INSERT`
BEFORE INSERT ON `Patient` FOR EACH ROW
BEGIN
      DECLARE DOB date;
    SELECT NEW.BirthDate INTO DOB;
    SET @dob = DOB;
     CALL ltc55311.IsOver18(@dob, @checkResult);
    IF @checkResult = 0
    THEN
    SIGNAL SQLSTATE '51000'
     SET MESSAGE_TEXT = 'Patient age is under 18!';
    END IF;
END

#Payment_BEFORE_UPDATE
CREATE DEFINER=`ltc55311`@`132.205.%.%` TRIGGER
`ltc55311`.`Payment_BEFORE_UPDATE`
BEFORE UPDATE ON `Payment` FOR EACH ROW
BEGIN
    declare appointmentStartTime datetime;
    Select Appointment.StartDateTime from ltc55311.Appointment
```

```
    WHERE Appointment.ID = NEW.PaymentStatus_ID

    AND OLD.PaymentStatus_ID != NEW.PaymentStatus_ID

    INTO appointmentStartTime;

    IF (NOW()>=STR_TO_DATE(appointmentStartTime, '%Y-%m-%d %H:%i:%s'))
THEN

          signal sqlstate '91000' set message_text = 'Appointment has
expired. The payment must be paid';

    END IF;

END


#Payment_AFTER_UPDATE

CREATE DEFINER=`ltc55311`@`132.205.%.%` TRIGGER
`ltc55311`.`Payment_AFTER_UPDATE`

AFTER UPDATE ON `Payment` FOR EACH ROW

BEGIN

     IF (NEW.PaymentMethod_ID!=4 && (NEW.PaymentStatus_ID != 1 &&
NEW.PaymentStatus_ID != 4))

    THEN

          # cannot change the value of current table after update
trigger

        signal sqlstate '92000' set message_text = 'PaymentStatus_ID
should be 1 (Paid)';

     END IF;

END


#Specialist_BEFORE_INSERT

CREATE DEFINER=`ltc55311`@`132.205.%.%` TRIGGER
`ltc55311`.`Specialist_BEFORE_INSERT`

BEFORE INSERT ON `Specialist` FOR EACH ROW

BEGIN

     DECLARE Type_ID INT(11);

     DECLARE Years INT(11);
```

```
    SELECT NEW.Employee_ID INTO @ID;
      SELECT NEW.SpecialistType_ID INTO @Type_ID;
      SELECT NEW.YearsExperience INTO @Years;

      CALL ltc55311.DocExperience(@Type_ID, @Years, @result);
      IF @result = 0
            THEN
            SIGNAL SQLSTATE '02000'
            SET MESSAGE_TEXT = 'Doc Years of experience must be 6
Terapist >2';
      END IF;
END

#Specialist_BEFORE_UPDATE
CREATE DEFINER=`ltc55311`@`132.205.%.%` TRIGGER
`ltc55311`.`Specialist_BEFORE_UPDATE`
BEFORE UPDATE ON `Specialist` FOR EACH ROW
BEGIN
            DECLARE sID INT(11);
    DECLARE Type_ID INT(11);
      DECLARE Years INT(11);
      SELECT NEW.Employee_ID INTO @sID;
    SELECT NEW.SpecialistType_ID INTO @Type_ID;
      SELECT NEW.YearsExperience INTO @Years;
      CALL ltc55311.DocExperience(@Type_ID, @Years, @result);
      IF @result = 0
            THEN
            SIGNAL SQLSTATE '02000'
            SET MESSAGE_TEXT = 'Doc Years of experience must be 6
Terapist >2';
      END IF;
```

```
    CALL ltc55311.DecreaseYears(@ID, @Years, @result);
    IF @result = 0
           THEN
           SIGNAL SQLSTATE '03000'
           SET MESSAGE_TEXT = 'Cannot decrease Years of Experience';

    END IF;
END

#Appointment_BEFORE_INSERT
CREATE DEFINER=`ltc55311`@`132.205.%.%` TRIGGER
`ltc55311`.`Appointment_BEFORE_INSERT`
 BEFORE INSERT ON `Appointment` FOR EACH ROW
BEGIN

    #Check if Appointment is being made with an active Prescription
      CALL IsActive(NEW.Prescription_ID, @result);
    IF ( @result = 0 )
    THEN
       SIGNAL SQLSTATE '41000'
      SET MESSAGE_TEXT = 'Inactive Prescription';
    END IF;

    #Check if Appoitment is being booked over 8 weeks in advance
    CALL DateValid(NEW.StartDateTime, @result);
    IF ( @result = 0)
    THEN
           SIGNAL SQLSTATE '42000'
        SET MESSAGE_TEXT = 'Cannot book over 8 weeks in advance';
    END IF;
```

```
    #Check if duration of appointment is exactly one hour
    CALL HourValid(NEW.StartDateTime, NEW.EndDateTime, @result);
     IF ( @result = 0)
    THEN
           SIGNAL SQLSTATE '43000'
        SET MESSAGE_TEXT = 'Appointment duration must be one hour';
    END IF;
    #Check if Patient doesn''t have any past owing payments
    CALL ZeroBalance(NEW.Patient_ID ,@result);
      IF ( @result = 0 )
           THEN
           SIGNAL SQLSTATE '44000'
           SET MESSAGE_TEXT = 'Balance owing';
     END IF;
     #Check if there is a scheduling conflict for specialist
     CALL SpecialistConflict(NEW.Prescription_ID,NEW.StartDateTime,
@result);

     IF ( @result = 0 )
           THEN
           SIGNAL SQLSTATE '45000'
           SET MESSAGE_TEXT = 'Specialist conflict';
     END IF;
     #Set creation date
     SET NEW.CreatedDateTime = CURDATE();
     #Set appointment status
     SET NEW.Status_ID = 1;

END

#Appointment_AFTER_INSERT
```

```
CREATE DEFINER=`ltc55311`@`132.205.%.%` TRIGGER
`ltc55311`.`Appointment_AFTER_INSERT`

AFTER INSERT ON `Appointment` FOR EACH ROW

BEGIN

     #After insert into Appointment, create an associated Payment
entry

INSERT INTO `ltc55311`.`Payment`

(

`Amount`,

`BillingDateTime`,

`PaymentStatus_ID`,

`Appointment_ID`)

VALUES (0, CURRENT_TIMESTAMP(), 2, NEW.ID);

END


#Appointment_BEFORE_UPDATE

CREATE DEFINER=`ltc55311`@`132.205.%.%` TRIGGER
`ltc55311`.`Appointment_BEFORE_UPDATE`

BEFORE UPDATE ON `Appointment` FOR EACH ROW

BEGIN

#Verify if field being updated is Appointment.Status_ID then otherwise
reject update

IF (NEW.ID != OLD.ID) OR (NEW.Patient_ID != OLD.Patient_ID) OR
(NEW.Prescription_ID != OLD.Prescription_ID)

OR (NEW.CreatedDateTime != OLD.CreatedDateTime) OR (NEW.StartDateTime
!= OLD.StartDateTime)

OR (NEW.EndDateTime != OLD.EndDateTime)

    THEN

       SIGNAL SQLSTATE '46000'

      SET MESSAGE_TEXT = 'Can only update "Status" field.';

END IF;

#If CURDATE() > Appointment.StartDateTime then status cannot be 3
(cancelled) or (booked).
```

```
IF (NOW() > OLD.StartDateTime) AND (NEW.Status_ID = 3 OR NEW.Status_ID
= 1)

     THEN

    SIGNAL SQLSTATE '47000'

    SET MESSAGE_TEXT = 'Appointment date has passed: cannot change
status to 3 (Cancelled) or 1 (Booked)';

END IF;

END


#Appointment_AFTER_UPDATE

CREATE DEFINER=`ltc55311`@`132.205.%.%` TRIGGER
`ltc55311`.`Appointment_AFTER_UPDATE`

AFTER UPDATE ON `Appointment` FOR EACH ROW

BEGIN

#If Appointment is cancelled, cancel the associated payment

IF NEW.Status_ID = 3

     THEN

          CALL ltc55311.CancelPayment(OLD.ID);

ELSE IF New.Status_ID = 1

#If Appointment is re-activated, activate the associated payment

     THEN

          CALL ltc55311.ActivatePayment(OLD.ID);

END IF;

END IF;

END
```

STORED PROCEDURES

```
#ActivatePayment

#changes payment status to pENDing

CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE `ActivatePayment`(IN
appointmentid INT(11))

BEGIN

#DECLARE EXIT HANDLER FOR SQLEXCEPTION SELECT 'Error occured';

UPDATE ltc55311.Payment

SET

Payment.PaymentStatus_ID = 2

WHERE Payment.Appointment_ID = appointmentid;

 END


#CancelPayment

#chages the payment status to cancelled after appointment cancellation

#UPDATE payment status to cancelled

CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE `CancelPayment`(IN
appointmentid INT(11))

BEGIN

#DECLARE EXIT HANDLER FOR SQLEXCEPTION SELECT 'Error occured';

UPDATE ltc55311.Payment

SET

Payment.PaymentStatus_ID = 4

WHERE Payment.Appointment_ID = appointmentid;

 END


#DateValid

#Ensures an appointment cannot be booked more than 8 weeks in advance

CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE `DateValid`(IN
StartDateTime DATETIME, OUT result BOOL )

BEGIN
```

```
DECLARE weeks INT(3);
SELECT ROUND(DATEDIFF(StartDateTime, CURDATE())/7,2)  INTO weeks;
IF  weeks > 8  THEN
     SET result = FALSE;
ELSE
     SET result = TRUE;
END IF;
END


#Ensures the employee years of experience cannot be reduced
CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE `DecreASeYears`(IN
specialist_id INT(11),Years INT(11), OUT result BOOL)
BEGIN
     SET result = FALSE;
     IF Years >
          (SELECT YearsExperience FROM ltc55311.Specialist
          WHERE ltc55311.Specialist.Employee_ID = specialist_id)
      THEN
          SET result = TRUE;
      END IF;
END


#DocExperience
#Ensures that a doctors minimum years of experience is 6 and a
therapist 2.
CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE `DocExperience`(IN
Type_ID INT(11),Years INT(11), OUT result BOOL)
BEGIN
     SET result = FALSE;
   IF Type_ID = 1 AND Years >= 6 THEN
```

```
            SET result = TRUE;
      ELSE IF Type_ID = 2 AND Years >= 2 THEN
            SET result = TRUE;
      END IF;
    END IF;
END
#GetAppointments
#Returns a patients appointments searchby userID
CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE `GetAppointments`(IN
id INT)
BEGIN
      SELECT a.ID, t.Name, p.PrescriptionNumber AS 'P#',
      a.CREATEdDateTIME, a.StartDateTime, a.EndDateTIME, s.Name AS
'AppointmentStatus'
    from Appointment a
            inner join Prescription p on a.Prescription_ID = p.ID
        inner join Treatment t on t.ID = p.Treatment_ID
            inner join Appointment_Status s on s.ID = a.Status_ID
        inner join Patient pat on pat.ID = a.Patient_ID
        inner join User u on u.ID = pat.User_ID

    WHERE
            u.ID = id;
  END
#GetPayments
#Returns payment information Associated with a patient by user ID
CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE `GetPayments`(IN id
INT)
BEGIN
      SELECT p.ID AS 'Payment_ID', p.Amount, p.BillingDateTIME,
a.Patient_ID,
```

```
    p.Appointment_ID, a.StartDateTime AS 'Appointment_StartTime',
pm.Name AS 'Pay_Method',
    ps.Name AS 'Pay_Status'
    from Payment p
           inner join Appointment a on p.Appointment_ID = a.ID
        inner join Patient pat on pat.ID = a.Patient_ID
        inner join User u on u.ID = pat.User_ID
        left join PaymentMethod pm on p.PaymentMethod_ID = pm.ID
        left join PaymentStatus ps on p.PaymentStatus_ID = ps.ID
    WHERE

           u.ID = id;
    END
#HourValid
#Ensures that every new appointment is exactly one hour long
CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE `HourValid`(IN
StartDateTime DATETIME, IN EndDateTIME DATETIME, OUT result BOOL)
BEGIN
     DECLARE hours DECIMAL(10);
     SELECT TIME_TO_SEC(TIMEDIFF(EndDateTIME, StartDateTime))/60 INTO
hours;
       IF (hours <> 60)  THEN
           SET result = FALSE;
     ELSE
           SET result = TRUE;
     END IF;
END
#IsActive
#Confirms that a prescription is active before an appointment can be
booked on the prescription ID
CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE `IsActive`(IN
prescription_id INT(11), OUT result BOOL)
```

```
BEGIN

DECLARE vCount INT(3);

    SET result = FALSE;

        SELECT count(Prescription.ID) INTO vCount FROM
ltc55311.Prescription

    WHERE Prescription.ID = prescription_id

    AND  Prescription.PrescriptionStatus_ID = 1;

        IF vCount = 1 THEN

           SET result = TRUE;

     END IF;

END

#IsOver18

#Ensures that any patient added to the system is at leASt 18 years old

CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE `IsOver18`(IN dob
date, OUT checkResult  BOOL )

BEGIN

    DECLARE age INT(3);

    SELECT TIMESTAMPDIFF(YEAR, dob, CURDATE()) INTO age;

        IF age > 17 THEN

           SET checkResult = TRUE;

     else

           SET checkResult = FALSE;

     END IF;

END

#SpecialistConflict

#Ensures that an appointments cannot overlap for any specialist

CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE
`SpecialistConflict`(IN prescription_ID INT(11), IN StartDateTime
DATETIME, OUT result BOOL)

BEGIN

DECLARE specialist_ID INT(11);

DECLARE total INT(3);
```

```
SELECT DISTINCT Prescription.Specialist_ID INTO specialist_ID FROM
ltc55311.Prescription WHERE Prescription.ID = prescription_ID;



SELECT count(Appointment.ID) INTO total FROM ltc55311.Appointment,
ltc55311.Prescription

WHERE Appointment.Prescription_ID = Prescription.ID

AND Appointment.StartDateTime = StartDateTime

AND Prescription.Specialist_ID = specialist_ID;

IF total > 0

THEN

     SET result = FALSE;

ELSE

     SET result = TRUE;

END IF;

END


#verifyCreditCards

#Populates a table with credit card information to be sent for
verification and sets payment status to paid

CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE
`verifyCreditCards`()

BEGIN

#INSERT INTO the CCVerification table the pENDing credit card payments

INSERT INTO `ltc55311`.`CCVerification`

(`ID`,

`PaymentDateTime`,

`VerificationDateTime`,

`Number`,

`ExpiryDate`,

`CVC`,

`Amount`)
```

```
SELECT Payment.ID, Payment.BillingDateTIME, CURRENT_TIMESTAMP(),
CreditCard.Number, CreditCard.ExpiryDate, CreditCard.CVC,
Payment.Amount

FROM ltc55311.Payment, ltc55311.CreditCard

WHERE

Payment.ID = CreditCard.Payment_ID

AND Payment.PaymentStatus_ID = 2

AND Payment.PaymentMethod_ID=4;

#Mark the payments AS verified

UPDATE `ltc55311`.`Payment`

SET

`PaymentStatus_ID` = 1

WHERE Payment.PaymentStatus_ID = 2

AND Payment.PaymentMethod_ID=4;

END


#ZeroBalance

#Confirms that a patient does not have an outstanding balance before
allowing them to make an appointment

CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE `ZeroBalance`(IN ID
INT(11),OUT result BOOL)

BEGIN

      DECLARE cc INT (4);

      SET result = TRUE;

    SELECT count(Appointment.ID) INTO cc

    FROM ltc55311.Appointment,ltc55311.Payment

    WHERE ltc55311.Appointment.Patient_ID = ID

    AND Payment.Appointment_ID = Appointment.ID

      AND Payment.PaymentStatus_ID = 3;

    IF cc > 0

           THEN

           SET result = FALSE;
```

```
    END IF;
END
#report1_NumPatientSeen
#Displays number of patients seen by each specialist over a given
period.

CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE
`report1_NumPatientSeen`( IN startDate DATETIME, IN ENDDate DATETIME)
BEGIN
SELECT
    concat(User.FirstName, ' ', User.LAStName) AS 'Specialist Name',
    count(Patient.ID) AS 'Patients Seen'
FROM
     ltc55311.Appointment,
    ltc55311.Prescription,
    ltc55311.Patient,
    ltc55311.Specialist,
    ltc55311.Employee,
    ltc55311.User
WHERE
     Prescription.ID = Appointment.Prescription_ID  AND
    Specialist.Employee_ID = Prescription.Specialist_ID AND
    Employee.ID = Specialist.Employee_ID AND
    User.ID = Employee.User_ID AND
    Prescription.Patient_ID = Patient.ID AND
    Appointment.StartDateTime >= startDate AND
    Appointment.EndDateTIME <= ENDDate

GROUP BY Prescription.Specialist_ID;

END
```

```
#report2_UnusedEquipment

#Returns a list of equipment that hAS not been used in a given TIME
frame

CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE
`report2_UnusedEquipment`

(IN startDate DATETIME, IN ENDDate DATETIME)

BEGIN

#Which piece of equipment hAS never been used?

SELECT DISTINCT *  FROM ltc55311.Equipment

WHERE Equipment.ID NOT IN


(SELECT DISTINCT Appointment_Equipment.Equipment_ID

FROM ltc55311.Appointment_Equipment, ltc55311.Appointment

WHERE Appointment.StartDateTime >= startDate

AND Appointment.EndDateTIME <= ENDDate

AND Appointment.ID = Appointment_Equipment.Appointment_ID

);

END


#report3_PatientsSeenAtCenter

#Returns information about all patients seen at a clinic

CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE
`report3_PatientsSeenAtCenter`(IN centerId INT)

BEGIN

# List all information available for physio patients who have been at
the center.

     SELECT Patient.ID, Patient.BirthDate, Patient.PhoneNumber,
Patient.InitialPrescription_ID, Patient.User_ID, Center.Name

     FROM
ltc55311.Patient,ltc55311.Appointment,ltc55311.Employee,ltc55311.Presc
ription, ltc55311.Center


     WHERE Patient.ID = Appointment.Patient_ID
```

```
      AND Appointment.status_ID = 2
      AND Appointment.Prescription_ID = Prescription.ID
      AND Prescription.Specialist_ID = Employee.ID
      AND Employee.Center_ID = Center.ID
      AND Center.ID = centerId;
END

#report4_TherapistAtCenter
#Returns information on therapists working at a clinic
CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE
`report4_TherapistAtCenter`(IN centerId INT)
BEGIN
#List all the information available for therapists who work at the
center.
SELECT DISTINCT
Employee.ID AS EmployeeID,
User.FirstName,
User.LastName,
SpecialistType.name AS SpecialistType
FROM
ltc55311.Center,
ltc55311.Employee,
ltc55311.User,
ltc55311.EmployeeType,
ltc55311.Specialist,
ltc55311.SpecialistType
WHERE Employee.Center_ID = centerId
AND Employee.EmployeeType_ID = 2
AND Employee.User_ID = User.ID
AND Employee.ID = Specialist.Employee_ID
AND Specialist.SpecialistType_ID = SpecialistType.ID
```

```
AND Employee.ID NOT IN
      (     SELECT DISTINCT Specialist.Employee_ID
            FROM ltc55311.Specialist, ltc55311.SpecialistType
                  WHERE Specialist.SpecialistType_ID = 1
                  );
END


#report 5 is similar to report 4 because we ASsumed that a specialist
can only work at once center
#report6_PatientAppointments
#Returns a patient's list of appointments'
CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE
`report6_PatientAppointments`(IN patientid INT(11))
BEGIN
#List details of reservations for a specific patient.
SELECT Appointment.ID AS AppID,
Appointment.Patient_ID,
User.LastName,
User.FirstName,
Appointment.Prescription_ID AS Presc,
Appointment.CREATEdDateTIME,
Appointment.StartDateTime AS TIME,
Appointment_Status.Name AS Status
FROM ltc55311.Appointment, ltc55311.Appointment_Status,
ltc55311.User,ltc55311.Patient
WHERE Appointment.Patient_ID = patientid
AND Appointment.Status_ID = Appointment_Status.ID
And Patient.ID=patientid
AND Patient.User_ID = User.ID;
END
```

```
#report7_DocAvailability

#Display's a doctor's availability in a given range of dates

CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE
`report7_DocAvailability`(IN specialist INT(11), IN date1 DATE, IN
date2 DATE)

BEGIN

INSERT INTO `ltc55311`.`DateTimeBag` (DateTimes)

#POPULATES TEMPORARY TABLE WITH ALL POSSIBLE APPOINTMENT TIMES FOR
GIVEN DATE DURATION

(SELECT TIMESTAMP(a.Date,Appointment_TIMEs.TIMEs)

from (

    SELECT CURDATE() - INTERVAL (a.a + (10 * b.a) + (100 * c.a)) DAY
AS Date

    FROM (SELECT 0 As a union all select 1 union all select 2 union
all select 3 union all select 4 union all select 5 union all select 6
union all select 7 union all select 8 union all select 9) AS a

    CROSS JOIN (SELECT 0 As a union all select 1 union all select 2
union all select 3 union all select 4 union all select 5 union all
select 6 union all select 7 union all select 8 union all select 9) AS
b

    CROSS JOIN (SELECT 0 As a union all select 1 union all select 2
union all select 3 union all select 4 union all select 5 union all
select 6 union all select 7 union all select 8 union all select 9) AS
c

) a, Appointment_TIMEs

WHERE a.Date between date1 and date2);

SELECT DateTimes FROM `ltc55311`.`DateTimeBag`

WHERE DateTimes NOT IN

#lists unavailability for physio therapist/doctor during a specified
period of TIME.

(SELECT DISTINCT
TIMESTAMP(DATE(Appointment.StartDateTime),TIME(Appointment.StartDateTi
me)) AS TIMES

FROM ltc55311.Appointment, ltc55311.Prescription, ltc55311.Specialist
```

```
WHERE Appointment.StartDateTime BETWEEN date1 and DATE_ADD(date2,
INTERVAL 1 DAY)

AND Appointment.Prescription_ID = Prescription.ID

AND Prescription.Specialist_ID = specialist);

#THE DIFFERENCE GIVES THE THERAPISTS AVAILABILITY

Truncate table `ltc55311`.`DateTimeBag`;

#CLEARS TEMPORARY TABLE

END


#reportA_Patient

#Report seen by patient. Displays a list of all prescriptions in the
patient`s record

CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE `reportA_Patient`(IN
userID INT (11))

BEGIN

#List prescriptions available for a patient..active and inactive

SELECT Patient_ID, User.LAStName, User.FirstName, Prescription.ID AS
PrescID,

 PrescriptionStatus.Description AS PrescStatus

FROM ltc55311.Prescription, ltc55311.Patient, ltc55311.User,
ltc55311.PrescriptionStatus

WHERE User.ID = userID

AND Patient.User_ID = User.ID

AND Prescription.Patient_ID = Patient.ID

AND Prescription.PrescriptionStatus_ID = PrescriptionStatus.ID;

END


#reportB_Doctor

#Report seen by doctors on patients with active prescriptions but no
future appointments and the

#date of the patient`s lASt appointment

CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE `reportB_Doctor`()
```

```
BEGIN

#List patients who have an active prescription and no appointments
booked

SELECT DISTINCT Prescription.ID AS PrescID, Prescription.Patient_ID,
User.LAStName,

User.FirstName, Patient.PhoneNumber, MAX(Appointment.StartDateTime) AS
LAStAppt

FROM ltc55311.Prescription, ltc55311.Appointment, ltc55311.Patient,
ltc55311.User

WHERE Prescription.PrescriptionStatus_ID = 1

AND Prescription.Patient_ID = Patient.ID

AND Patient.User_ID = User.ID

AND Prescription.ID = Appointment.Prescription_ID

AND Appointment.ID NOT IN(

      SELECT Appointment.ID

    FROM ltc55311.Appointment

    WHERE Appointment.StartDateTime >= curdate()

)

GROUP BY Prescription.ID ;

END


#reportC_Therapist

#Report seen by therapists on equipment use frequency in the lASt x
days

CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE
`reportC_Therapist`(IN days INT(5), IN frequency INT(5))

BEGIN

#List equipment used more than x TIMEs in the x days

DECLARE period DATETIME;

SELECT SUBDATE(CURDATE(), INTERVAL days DAY) INTO period;

SELECT count(Appointment_Equipment.ID) AS used, Equipment.Name

FROM ltc55311.Appointment, ltc55311.Appointment_Equipment,Equipment
```

```
WHERE Appointment.StartDateTime >= period
AND Appointment.ID = Appointment_Equipment.Appointment_ID
AND Appointment_Equipment.Equipment_ID = Equipment.ID
GROUP BY Appointment_Equipment.Equipment_ID
having used >frequency;
END


#reportD_Receptionist
#Report seen by receptioninst. Shows the list of patients in a given
age range
CREATE DEFINER=`ltc55311`@`132.205.%.%` PROCEDURE
`reportD_Receptionist`(IN age1 INT (5), IN age2 INT(5))
BEGIN
#Patient count grouped by age group specified
SELECT Patient.ID, User.FirstName, User.LAStName, Patient.BirthDate,
Patient.PhoneNumber, Patient.InitialPrescription_ID    AS PrescID
FROM ltc55311.Patient, ltc55311.User
WHERE TIMESTAMPDIFF(YEAR, Patient.BirthDate, CURDATE()) > age1
AND TIMESTAMPDIFF(YEAR, Patient.BirthDate, CURDATE()) < age2
AND Patient.User_ID = User.ID;
END
```

ROUTINES

```
DELIMITER |

CREATE EVENT ltc55311VerificationScheduler
    ON SCHEDULE EVERY 1 MONTH STARTS '2017-08-01 18:30:00'
    DO
      BEGIN
        #Insert into the CCVerification table the pending credit card
payments
INSERT INTO `ltc55311`.`CCVerification`
(`ID`,
`PaymentDateTime`,
`VerificationDateTime`,
`Number`,
`ExpiryDate`,
`CVC`,
`Amount`)
SELECT Payment.ID, Payment.BillingDateTime, CURRENT_TIMESTAMP(),
CreditCard.Number, CreditCard.ExpiryDate, CreditCard.CVC,
Payment.Amount
FROM ltc55311.Payment, ltc55311.CreditCard
WHERE
Payment.ID = CreditCard.Payment_ID
AND Payment.PaymentStatus_ID = 2
AND Payment.PaymentMethod_ID=4;


#Mark the payments as verified
UPDATE `ltc55311`.`Payment`
SET
`PaymentStatus_ID` = 1
WHERE Payment.PaymentStatus_ID = 2
AND Payment.PaymentMethod_ID=4;
      END |

DELIMITER ;


DELIMITER |

CREATE EVENT changeAppointmentStatus
    ON SCHEDULE EVERY 1 MONTH STARTS '2017-08-05 08:00:00'
    DO
      BEGIN
        #Insert into the CCVerification table the pending credit card
payments
```

```
UPDATE `ltc55311`.`Appointment`
SET
Status_ID =     2
WHERE Appointment.Status_ID = 1
And Appoitment.ID =

           (select ID
           from ltc55311.Appointment
           where Appointment.StartDateTime = curdate());

      END |

DELIMITER ;
```

SECTION 3.1

PHP SCRIPT

Please see attached CD

SECTION 3.2

REPORTS

Suggested Reports

1. How many patients has each physio therapists seen in a specified period of time?
2. Which piece of equipment has never been used?
3. List all information available for physio patients who have been at the center.
4. List all the information available for therapists who have been at the center.
5. List all the information available for therapists who work at the center.
6. List detail of reservations for a specific patient.
7. List availability for physio therapist/doctor during a specified period of time.

Assumption: Reports 4 and 5 are identical as therapists are intended to work for only one center

Additional reports accessible only to specified user type

1. Receptionist - Patient list and count by age group
2. Patient - Prescription report
3. Doctor - List of patients with active prescriptions but no future appointments
4. Therapist - Frequently used equipment