FITDAY SYSTEM

INFO 6210 SECTION 04

TEAM NAME: DATA REVOLVERS

TEAM MEMBERS: SUMIT DESHMUKH, LEANDRA MENEZES, MAANSI CHANDIRA, VIPRA SHAH

Table of Contents

1. INTRODUCTION

Over 21 million people of the population are suffering from diabetes in USA alone. More than 8 million people have diabetes, yet unaware about it. Common symptoms of diabetes: Cuts/bruises that are slow to heal, Feeling very hungry - even though you are eating. Current diagnosing technique involves invasive blood tests such as Random blood sugar test Fasting blood sugar test. Necessity of noninvasive technique to continuously monitor glucose is the need of the hour. FitDay foresees that fitness tracker devices will include biosensors to monitor a Diabetic patient in the near future. Having glucose information as easy as heart rate or calories burned will be vital to live a healthy life.

2. USE CASES

The FitDay system identifies three main use cases:

1. FitDay System User

User registers himself in the FitDay system as **Athlete** and/or **Patient**. The system stores diabetic measures or activity data from user's device. Analysis of User Data is done and appropriate suggestions are made.

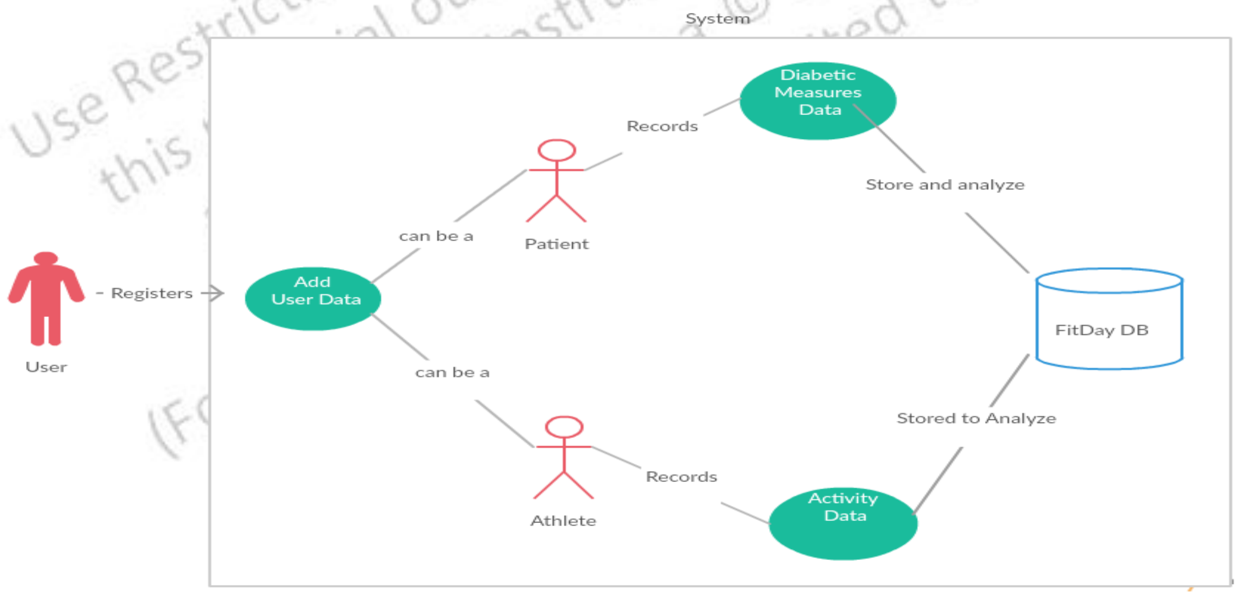


Figure 1: User Interaction

2. Doctor Patient

FitDay system after processing diabetic measures of each patient provides suggestion to doctor. Doctor refers to the system suggestion and prescribes treatment to the patient

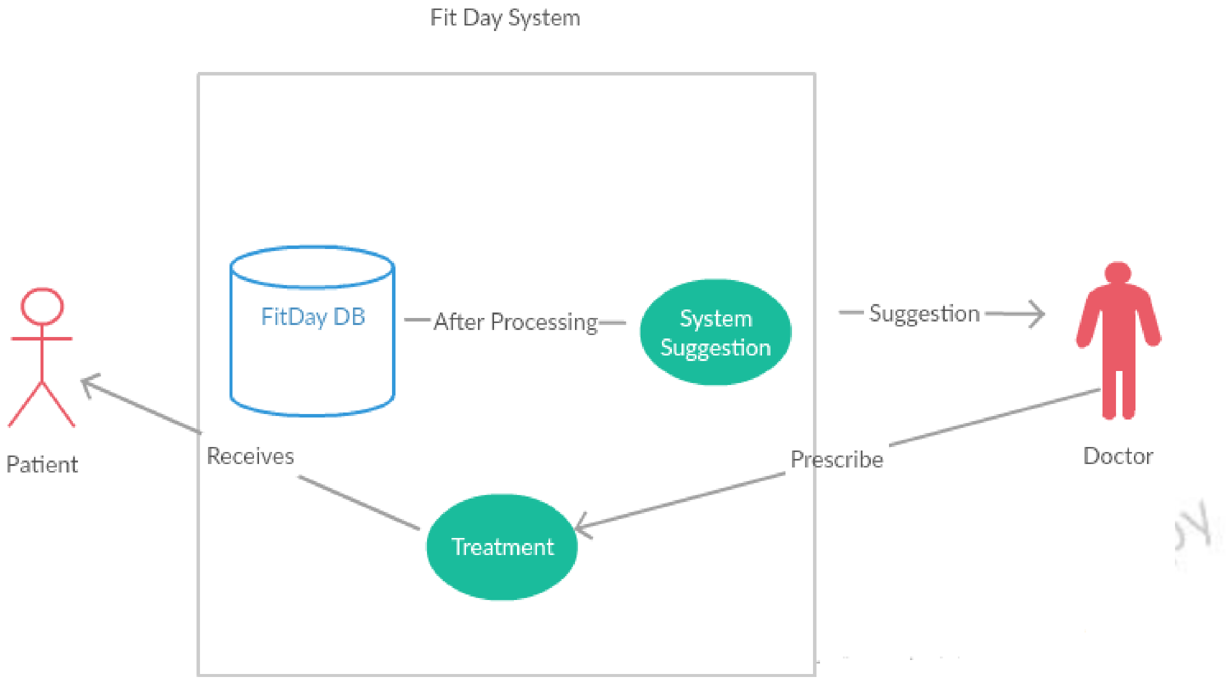


Figure 2 Doctor Interaction

3. Athlete

Athlete activity and vital signs are recorded in the system and stored in the FitDay database for analysis. The system analyzes heart rate and suggests remedies to the athlete. Calories burned during the activities are also stored for each athlete.

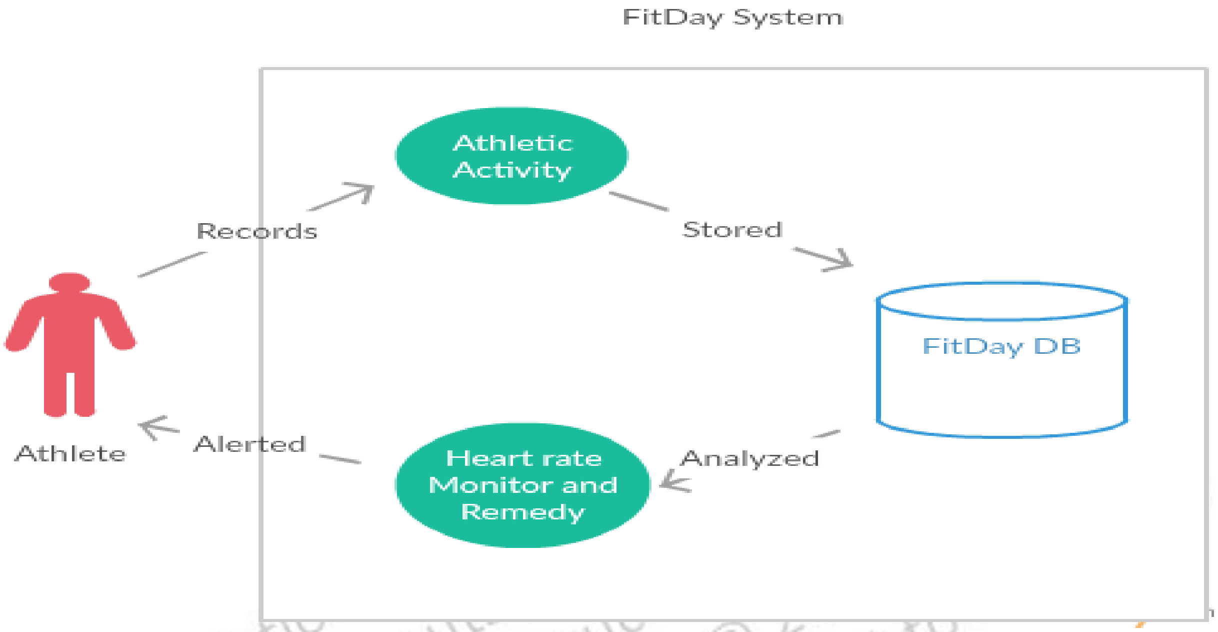
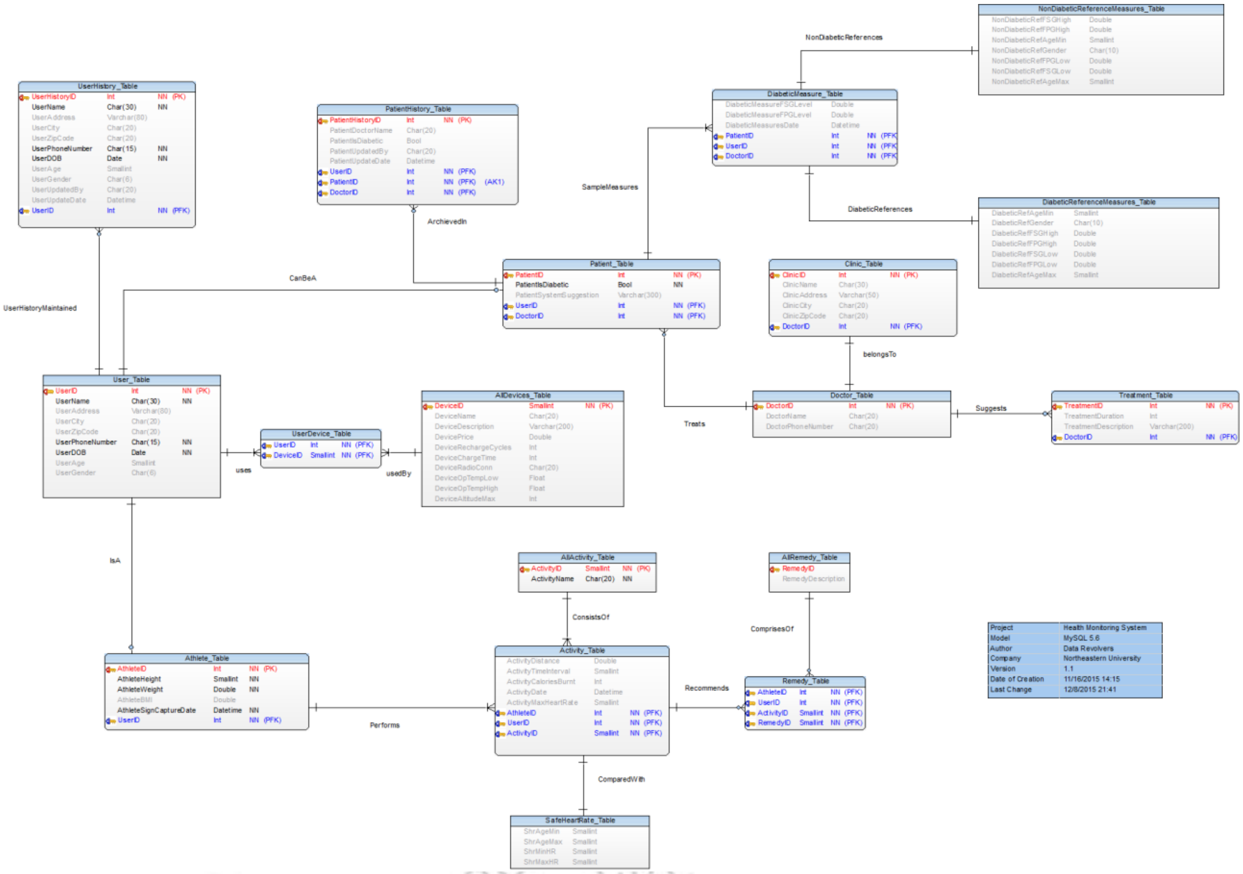


Figure 3 Athlete Interaction

3. ENHANCED ENTITY RELATIONSHIP DIAGRAM

An EER diagram was made for the entire database, encompassing all the above mentioned roles and other relevant data sets.



UserHistory_Table

UserHistoryID	Int	NN	(PK)
UserName	Char(30)	NN	
UserAddress	Varchar(80)		
UserCity	Char(20)		
UserZipCode	Char(20)		
UserPhoneNumber	Char(15)	NN	
UserDOB	Date	NN	
UserAge	Smallint		
UserGender	Char(6)		
UserUpdatedBy	Char(20)		
UserUpdateDate	Datetime		
UserID	Int	NN	(PFK)

PatientHistory_Table

PatientHistoryID	Int	NN	(PK)	
PatientDoctorName	Char(20)			
PatientIsDiabetic	Bool			
PatientUpdatedBy	Char(20)			
PatientUpdateDate	Datetime			
UserID	Int	NN	(PFK)	
PatientID	Int	NN	(PFK)	(AK1)
DoctorID	Int	NN	(PFK)	

DiabeticMeasure_Table

DiabeticMeasureFSGLevel	Double		
DiabeticMeasureFPGLevel	Double		
DiabeticMeasuresDate	Datetime		
PatientID	Int	NN	(PFK
UserID	Int	NN	(PFK
DoctorID	Int	NN	(PFK

NonDiabeticReferenceMeasures_Table

NonDiabeticRefFSGHigh	Double
NonDiabeticRefFPGHigh	Double
NonDiabeticRefAgeMin	Smallint
NonDiabeticRefGender	Char(10)
NonDiabeticRefFPGLow	Double
NonDiabeticRefFSGLow	Double
NonDiabeticRefAgeMax	Smallint

DiabeticReferenceMeasures_Table

DiabeticRefAgeMin	Smallint
DiabeticRefGender	Char(10)
DiabeticRefFSGHigh	Double
DiabeticRefFPGHigh	Double
DiabeticRefFSGLow	Double
DiabeticRefFPGLow	Double
DiabeticRefAgeMax	Smallint

Patient_Table

PatientID	Int	NN	(PK)
PatientIsDiabetic	Bool	NN	
PatientSystemSuggestion	Varchar(300)		
UserID	Int	NN	(PFK)
DoctorID	Int	NN	(PFK)

Clinic_Table

ClinicID	Int	NN	(PK)
ClinicName	Char(30)		
ClinicAddress	Varchar(50)		
ClinicCity	Char(20)		
ClinicZipCode	Char(20)		
DoctorID	Int	NN	(PFK)

User_Table

UserID	Int	NN	(PK)
UserName	Char(30)	NN	
UserAddress	Varchar(80)		
UserCity	Char(20)		
UserZipCode	Char(20)		
UserPhoneNumber	Char(15)	NN	
UserDOB	Date	NN	
UserAge	Smallint		
UserGender	Char(6)		

UserDevice_Table

UserID	Int	NN	(PFK)
DeviceID	Smallint	NN	(PFK)

AllDevices_Table

DeviceID	Smallint	NN	(PK)
DeviceName	Char(20)		
DeviceDescription	Varchar(200)		
DevicePrice	Double		
DeviceRechargeCycles	Int		
DeviceChargeTime	Int		
DeviceRadioConn	Char(20)		
DeviceOpTempLow	Float		
DeviceOpTempHigh	Float		
DeviceAltitudeMax	Int		

Doctor_Table

DoctorID	Int	NN	(PK)
DoctorName	Char(20)		
DoctorPhoneNumber	Char(20)		

Treatment_Table

TreatmentID	Int	NN	(PK)
TreatmentDuration	Int		
TreatmentDescription	Varchar(200)		
DoctorID	Int	NN	(PFK)

AllActivity_Table

ActivityID	Smallint	NN	(PK)
ActivityName	Char(20)	NN	

AllRemedy_Table

RemedyID
RemedyDescription

Athlete_Table

AthleteID	Int	NN	(PK)
AthleteHeight	Smallint	NN	
AthleteWeight	Double	NN	
AthleteBMI	Double		
AthleteSignCaptureDate	Datetime	NN	
UserID	Int	NN	(PFK)

Activity_Table

ActivityDistance	Double		
ActivityTimeInterval	Smallint		
ActivityCaloriesBurnt	Int		
ActivityDate	Datetime		
ActivityMaxHeartRate	Smallint		
AthleteID	Int	NN	(PFK)
UserID	Int	NN	(PFK)
ActivityID	Smallint	NN	(PFK)

Remedy_Table

AthleteID	Int	NN	(PFK)
UserID	Int	NN	(PFK)
ActivityID	Smallint	NN	(PFK)
RemedyID	Smallint	NN	(PFK)

SafeHeartRate_Table

ShrAgeMin	Smallint
ShrAgeMax	Smallint
ShrMinHR	Smallint
ShrMaxHR	Smallint

Project	Health Monitoring System
Model	MySQL 5.6
Author	Data Revolvers
Company	Northeastern University
Version	1.1
Date of Creation	11/16/2015 14:15
Last Change	12/8/2015 21:41

Figure 4 EER Diagram

3.1 NORMALIZATION

As with most databases, this database aimed to achieve the third normal form (3NF) level of normalization for all tables. The definition of the 3NF normalization is that all the attributes in a table are directly related to only the primary key and no other attribute.

Normalized databases fair very well under conditions where the applications are write-intensive and the write-load is more than the read-load.Main cause of concern with fully normalized tables is that normalized data means joins between tables. Joining means that read operations have to suffer because indexing strategies do not go well with table joins.

3.2 DE-NORMALIZATION

Denormalized databases fair well under heavy read-load and when the application is read intensive. The data is present in the same table so there is no need for any joins, hence the selects are very fast.A single table with all the required data allows much more efficient index usage. Selects can be very fast on denormalized tables, but because the data is duplicated, the updates and inserts become complex and costly. Denormalized schema greatly improves performance with efficient use of triggers.

4. ASSUMPTIONS MADE FOR THE DATABASE

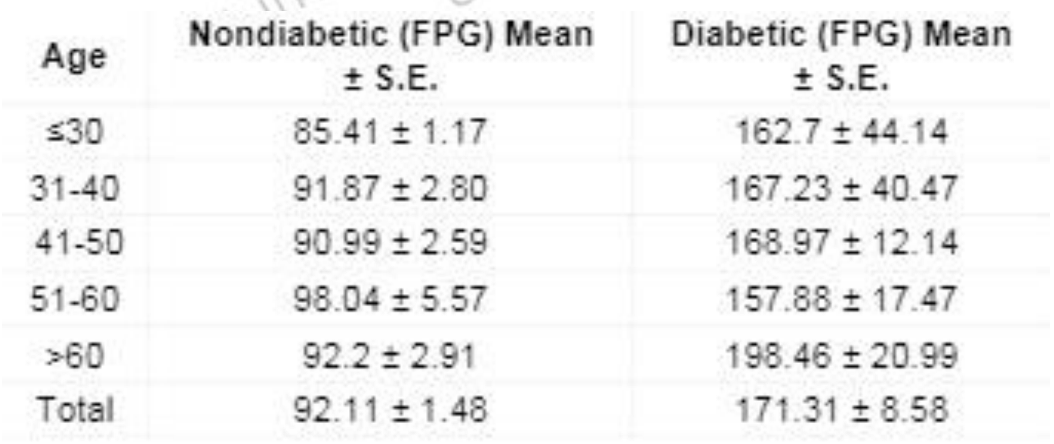
- Data isn't 100% factual – interpolated and extrapolated with existing data.
- Patients have already been diagnosed with diabetes and doctor is monitoring patient through FitDay system.
- Continuous diabetic measures are being collected from patients and fed into the FitDay system.
- FitDay system gives suggestion to doctors and doctor prescribes appropriate treatment accordingly. Athletes are recording their activity details in the FitDay system.

5. CALCULATION PROCEDURE

For running activity on flat surface: CB = [0.0215 x KPH^3 - 0.1765 x KPH^2 + 0.8710 x KPH + 1.4577] x WKG x T

CB = Calorie burn (in calories)
KPH = Running speed (in kilometers per hour)
WKG = Weight (in kilograms)
T = Time (in hours)

For Diabetic patients:

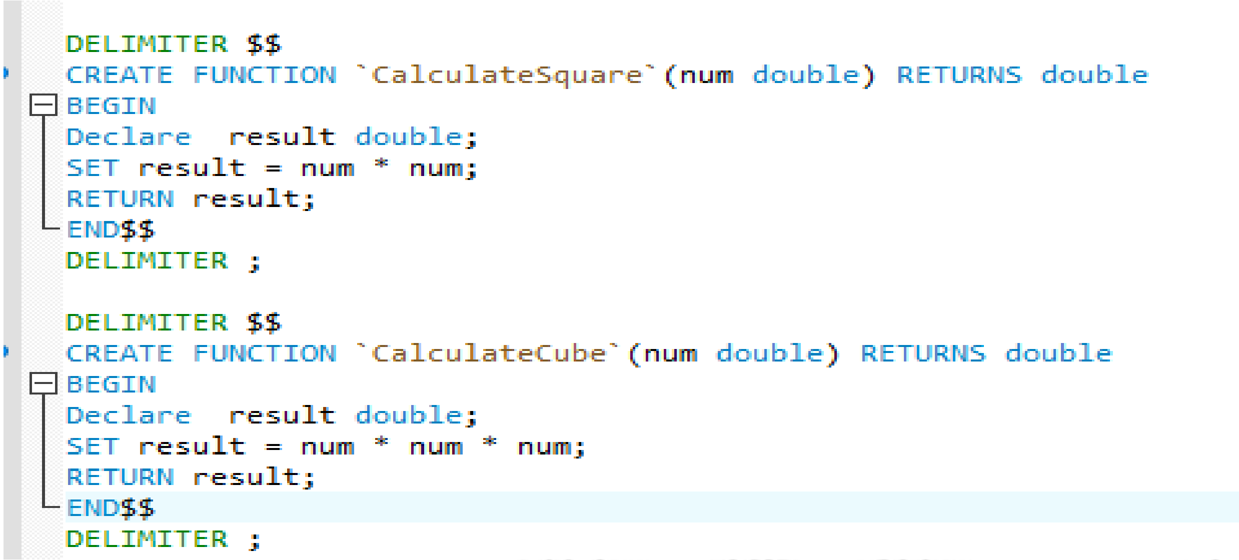


Age	Nondiabetic (FPG) Mean ± S.E.	Diabetic (FPG) Mean ± S.E.
≤30	85.41 ± 1.17	162.7 ± 44.14
31-40	91.87 ± 2.80	167.23 ± 40.47
41-50	90.99 ± 2.59	168.97 ± 12.14
51-60	98.04 ± 5.57	157.88 ± 17.47
>60	92.2 ± 2.91	198.46 ± 20.99
Total	92.11 ± 1.48	171.31 ± 8.58

Table 1 Diabetic Patient FPG and FSG

6. .FUNCTIONS

Functions to facilitate Calories Burned calculation



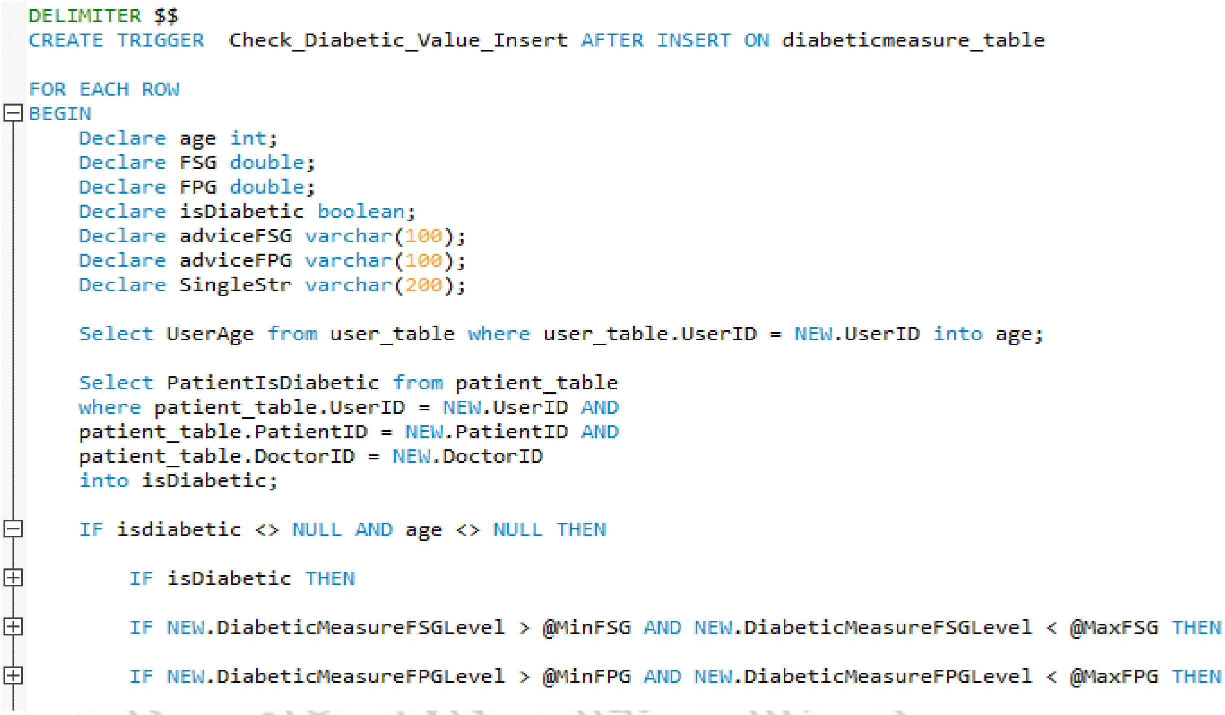
```
DELIMITER $$
CREATE FUNCTION `CalculateSquare`(num double) RETURNS double
BEGIN
Declare  result double;
SET result = num * num;
RETURN result;
END$$
DELIMITER ;

DELIMITER $$
CREATE FUNCTION `CalculateCube`(num double) RETURNS double
BEGIN
Declare  result double;
SET result = num * num * num;
RETURN result;
END$$
DELIMITER ;
```

Screenshot 1 Square and Cube Function

7. TRIGGERS

Database trigger is powerful tool for protecting the integrity of the data in your MySQL databases. Database triggers are very useful to automate some database operations such as audit logging.

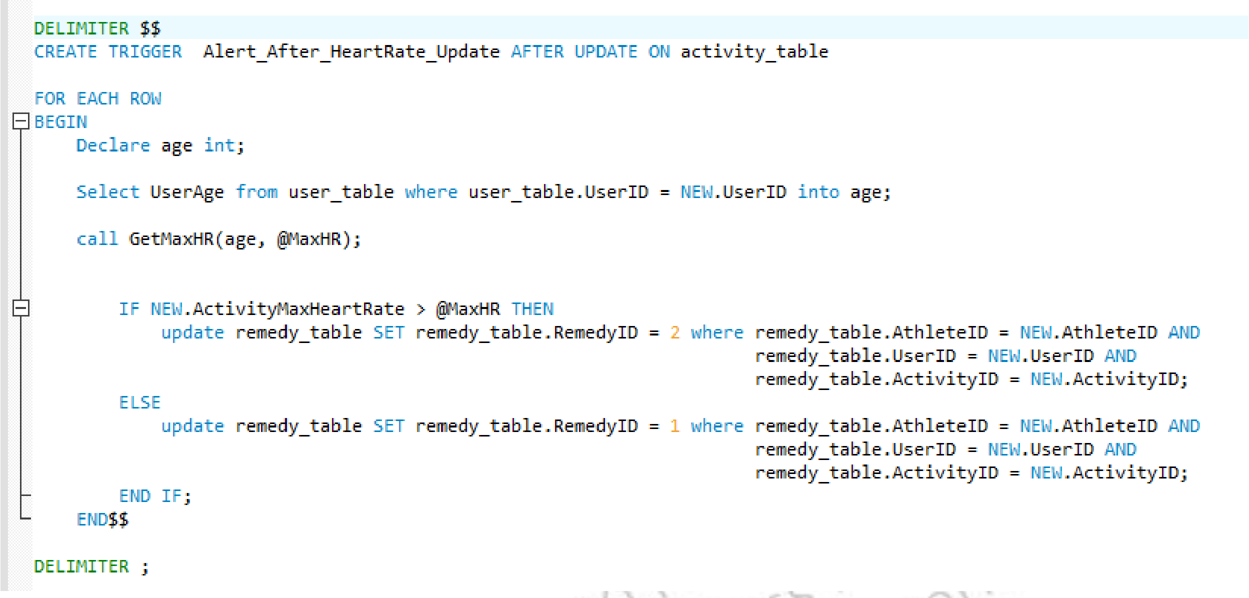
- Insert Trigger to suggest doctor the Level of FSG & FPG in a patient



```
DELIMITER $$
CREATE TRIGGER  Check_Diabetic_Value_Insert AFTER INSERT ON diabeticmeasure_table

FOR EACH ROW
BEGIN
    Declare age int;
    Declare FSG double;
    Declare FPG double;
    Declare isDiabetic boolean;
    Declare adviceFSG varchar(100);
    Declare adviceFPG varchar(100);
    Declare SingleStr varchar(200);

    Select UserAge from user_table where user_table.UserID = NEW.UserID into age;

    Select PatientIsDiabetic from patient_table
    where patient_table.UserID = NEW.UserID AND
    patient_table.PatientID = NEW.PatientID AND
    patient_table.DoctorID = NEW.DoctorID
    into isDiabetic;

    IF isdiabetic <> NULL AND age <> NULL THEN

        IF isDiabetic THEN

        IF NEW.DiabeticMeasureFSGLevel > @MinFSG AND NEW.DiabeticMeasureFSGLevel < @MaxFSG THEN

        IF NEW.DiabeticMeasureFPGLevel > @MinFPG AND NEW.DiabeticMeasureFPGLevel < @MaxFPG THEN
```

Screenshot 2 Trigger Check Diabetic value Insert

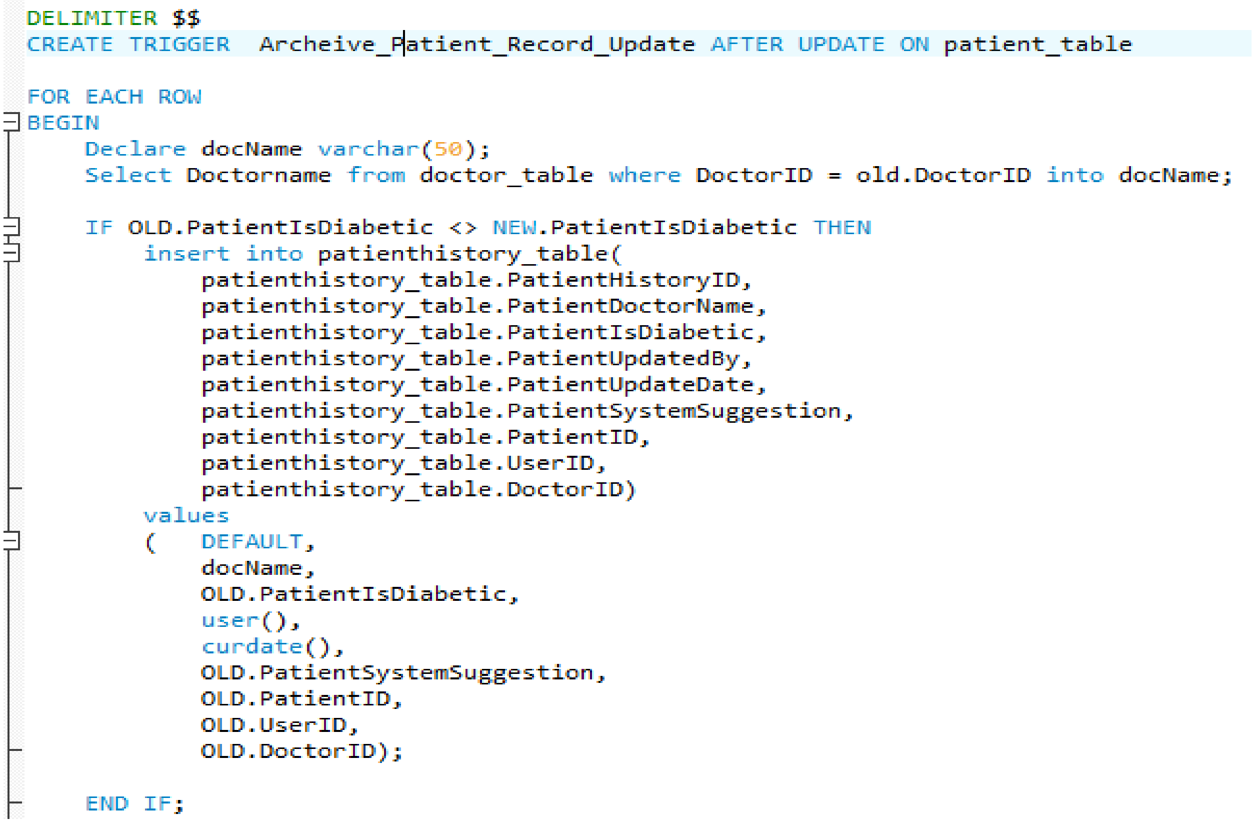
- Update Trigger to suggest Athlete the Remedy after a given activity



```
DELIMITER $$
CREATE TRIGGER  Alert_After_HeartRate_Update AFTER UPDATE ON activity_table

FOR EACH ROW
BEGIN
    Declare age int;

    Select UserAge from user_table where user_table.UserID = NEW.UserID into age;

    call GetMaxHR(age, @MaxHR);


        IF NEW.ActivityMaxHeartRate > @MaxHR THEN
            update remedy_table SET remedy_table.RemedyID = 2 where remedy_table.AthleteID = NEW.AthleteID AND
                                                                    remedy_table.UserID = NEW.UserID AND
                                                                    remedy_table.ActivityID = NEW.ActivityID;
        ELSE
            update remedy_table SET remedy_table.RemedyID = 1 where remedy_table.AthleteID = NEW.AthleteID AND
                                                                    remedy_table.UserID = NEW.UserID AND
                                                                    remedy_table.ActivityID = NEW.ActivityID;

        END IF;
    END$$

DELIMITER ;
```

Screenshot 3 ALERT AFTER HEART RATE UPDATE

- Trigger to archive patient data when patient’s diabetic state is changed

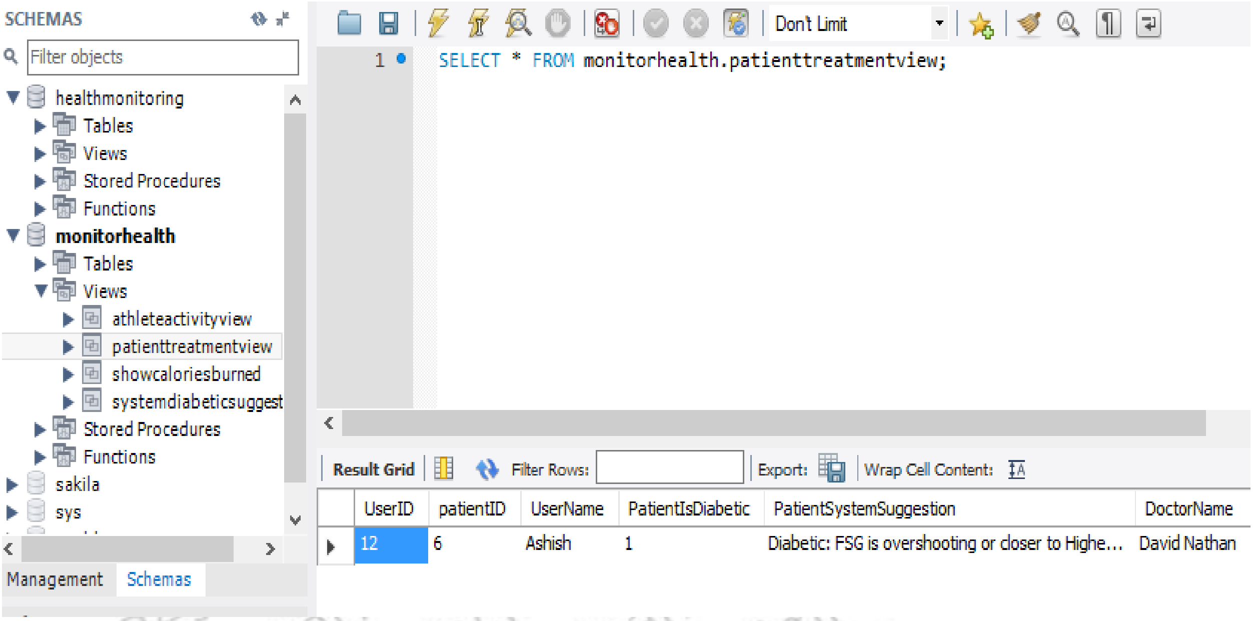


```
DELIMITER $$
CREATE TRIGGER  Archeive_Patient_Record_Update AFTER UPDATE ON patient_table

FOR EACH ROW
BEGIN
    Declare docName varchar(50);
    Select Doctorname from doctor_table where DoctorID = old.DoctorID into docName;

    IF OLD.PatientIsDiabetic <> NEW.PatientIsDiabetic THEN
        insert into patienthistory_table(
            patienthistory_table.PatientHistoryID,
            patienthistory_table.PatientDoctorName,
            patienthistory_table.PatientIsDiabetic,
            patienthistory_table.PatientUpdatedBy,
            patienthistory_table.PatientUpdateDate,
            patienthistory_table.PatientSystemSuggestion,
            patienthistory_table.PatientID,
            patienthistory_table.UserID,
            patienthistory_table.DoctorID)
        values
        (   DEFAULT,
            docName,
            OLD.PatientIsDiabetic,
            user(),
            curdate(),
            OLD.PatientSystemSuggestion,
            OLD.PatientID,
            OLD.UserID,
            OLD.DoctorID);

    END IF;
```

Screenshot 4 Archive Patient Record Update

8. VIEWS

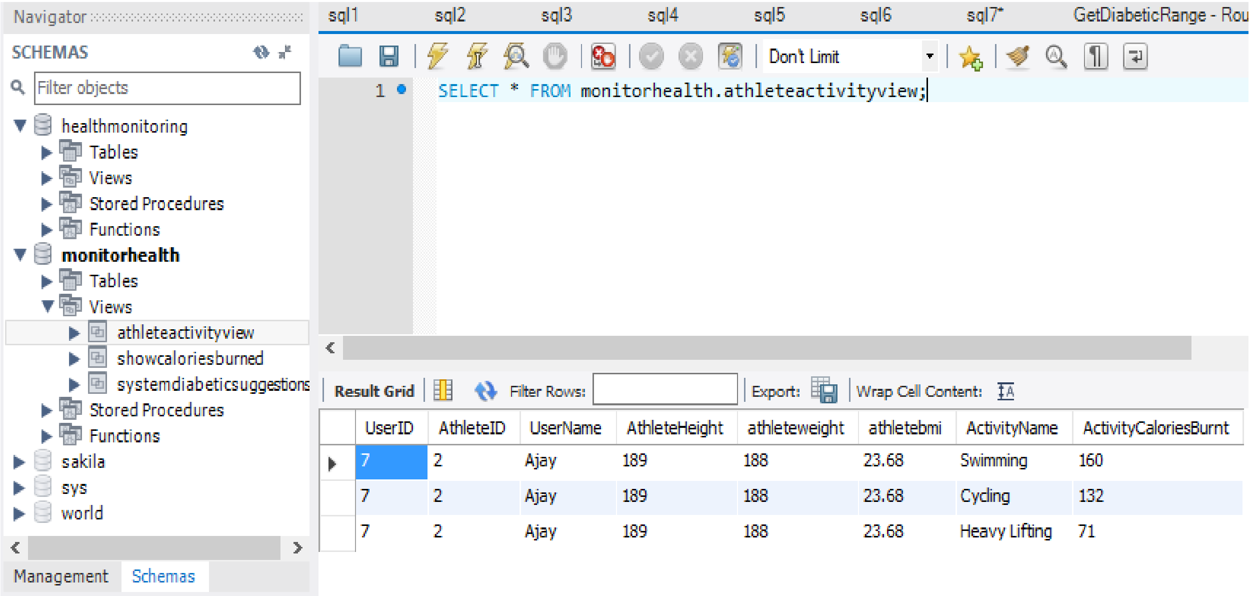
A database view is known as a “virtual table” that allows you to query the data in it. Understanding database views and using them correctly are very important. In this section, we will discuss about the database views, how they are implemented in MySQL, and how to use them more effectively.



```
SELECT * FROM monitorhealth.patienttreatmentview;
```

UserID	patientID	UserName	PatientIsDiabetic	PatientSystemSuggestion	DoctorName
12	6	Ashish	1	Diabetic: FSG is overshooting or closer to Highe...	David Nathan

Screenshot 5Patient Treatment View

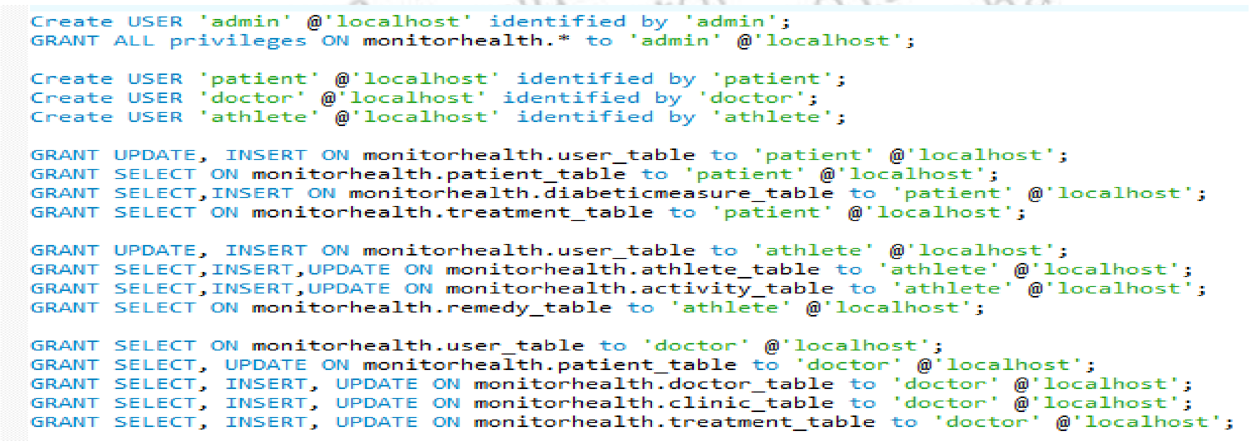


```
SELECT * FROM monitorhealth.athleteactivityview;
```

UserID	AthleteID	UserName	AthleteHeight	athleteweight	athletebmi	ActivityName	ActivityCaloriesBurnt
7	2	Ajay	189	188	23.68	Swimming	160
7	2	Ajay	189	188	23.68	Cycling	132
7	2	Ajay	189	188	23.68	Heavy Lifting	71

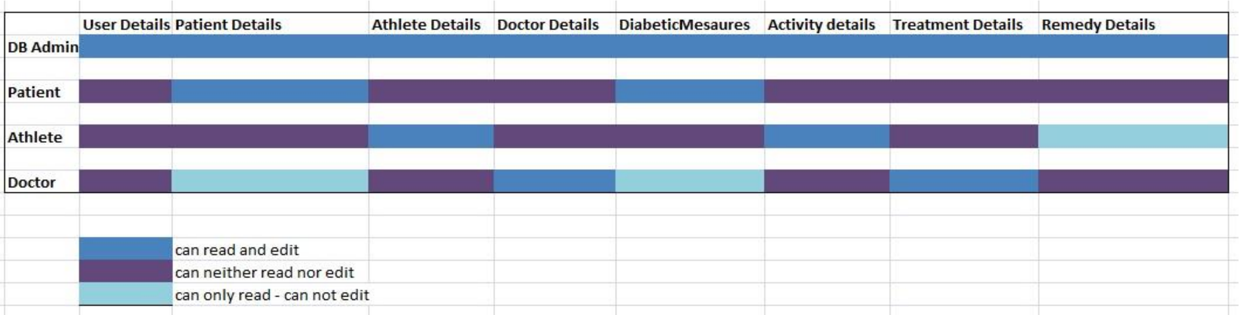
Screenshot 6 athlete activity view

9. USERS AND PRIVILEGES



```
Create USER 'admin' @'localhost' identified by 'admin';
GRANT ALL privileges ON monitorhealth.* to 'admin' @'localhost';

Create USER 'patient' @'localhost' identified by 'patient';
Create USER 'doctor' @'localhost' identified by 'doctor';
Create USER 'athlete' @'localhost' identified by 'athlete';

GRANT UPDATE, INSERT ON monitorhealth.user_table to 'patient' @'localhost';
GRANT SELECT ON monitorhealth.patient_table to 'patient' @'localhost';
GRANT SELECT,INSERT ON monitorhealth.diabeticmeasure_table to 'patient' @'localhost';
GRANT SELECT ON monitorhealth.treatment_table to 'patient' @'localhost';

GRANT UPDATE, INSERT ON monitorhealth.user_table to 'athlete' @'localhost';
GRANT SELECT,INSERT,UPDATE ON monitorhealth.athlete_table to 'athlete' @'localhost';
GRANT SELECT,INSERT,UPDATE ON monitorhealth.activity_table to 'athlete' @'localhost';
GRANT SELECT ON monitorhealth.remedy_table to 'athlete' @'localhost';

GRANT SELECT ON monitorhealth.user_table to 'doctor' @'localhost';
GRANT SELECT, UPDATE ON monitorhealth.patient_table to 'doctor' @'localhost';
GRANT SELECT, INSERT, UPDATE ON monitorhealth.doctor_table to 'doctor' @'localhost';
GRANT SELECT, INSERT, UPDATE ON monitorhealth.clinic_table to 'doctor' @'localhost';
GRANT SELECT, INSERT, UPDATE ON monitorhealth.treatment_table to 'doctor' @'localhost';
```

Screenshot 7User and Privileges



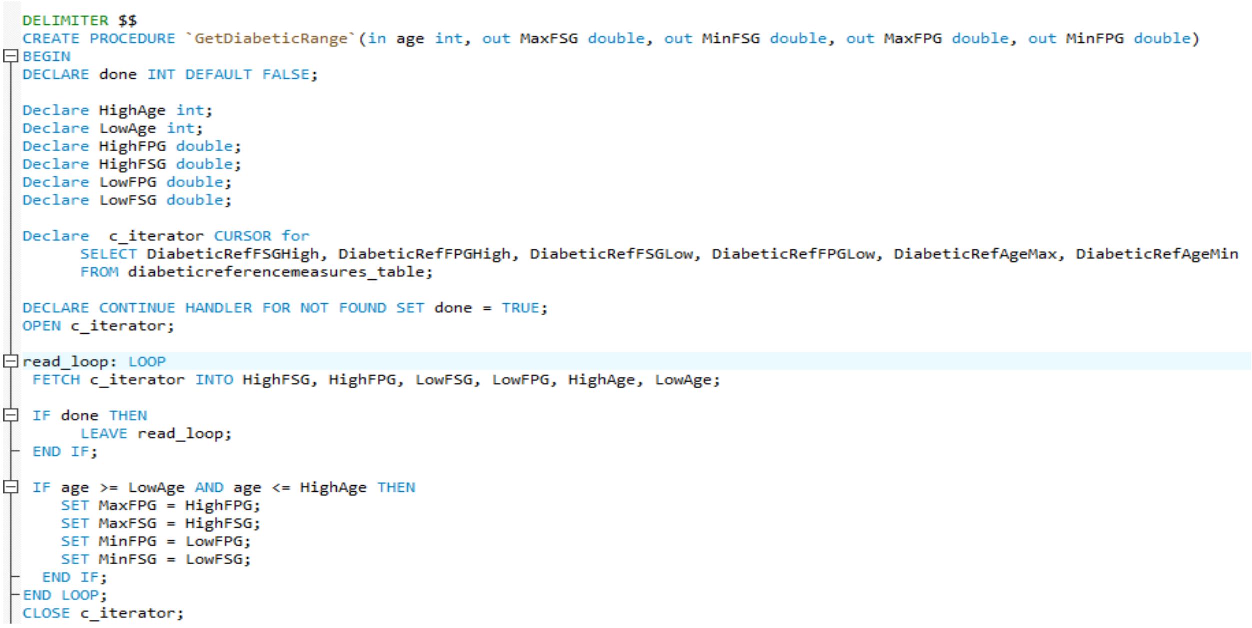
	User Details	Patient Details	Athlete Details	Doctor Details	DiabeticMesaures	Activity details	Treatment Details	Remedy Details
DB Admin	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
Patient	🟪	🟦	🟪	🟪	🟦	🟪	🟪	🟪
Athlete	🟪	🟪	🟦	🟪	🟪	🟦	🟪	▨
Doctor	🟪	▨	🟪	🟦	▨	🟪	🟦	🟪

🟦	can read and edit
🟪	can neither read nor edit
▨	can only read - can not edit

Screenshot 8Table Access

10. STORED PROCEDURES

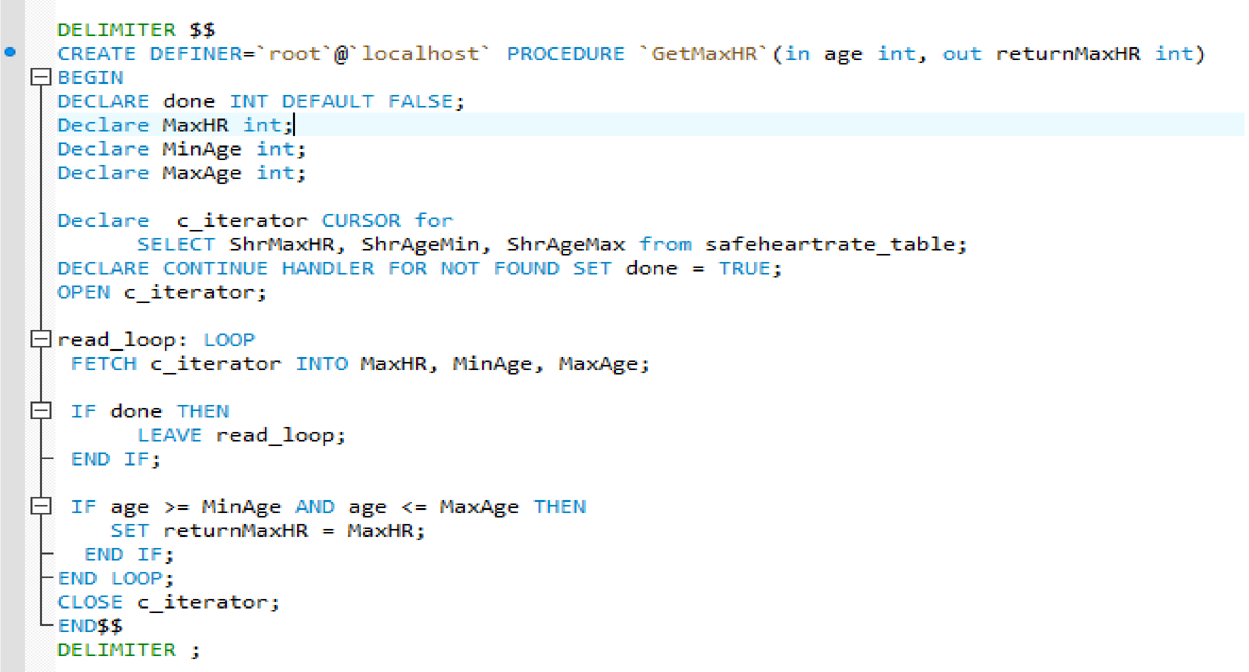
Get Diabetic Range for particular age group



```
DELIMITER $$
CREATE PROCEDURE `GetDiabeticRange`(in age int, out MaxFSG double, out MinFSG double, out MaxFPG double, out MinFPG double)
BEGIN
DECLARE done INT DEFAULT FALSE;

Declare HighAge int;
Declare LowAge int;
Declare HighFPG double;
Declare HighFSG double;
Declare LowFPG double;
Declare LowFSG double;

Declare  c_iterator CURSOR for
      SELECT DiabeticRefFSGHigh, DiabeticRefFPGHigh, DiabeticRefFSGLow, DiabeticRefFPGLow, DiabeticRefAgeMax, DiabeticRefAgeMin
      FROM diabeticreferencemeasures_table;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
OPEN c_iterator;

read_loop: LOOP
 FETCH c_iterator INTO HighFSG, HighFPG, LowFSG, LowFPG, HighAge, LowAge;

 IF done THEN
      LEAVE read_loop;
 END IF;

 IF age >= LowAge AND age <= HighAge THEN
    SET MaxFPG = HighFPG;
    SET MaxFSG = HighFSG;
    SET MinFPG = LowFPG;
    SET MinFSG = LowFSG;
  END IF;
END LOOP;
CLOSE c_iterator;
```

Screenshot 9Get Diabetic Range

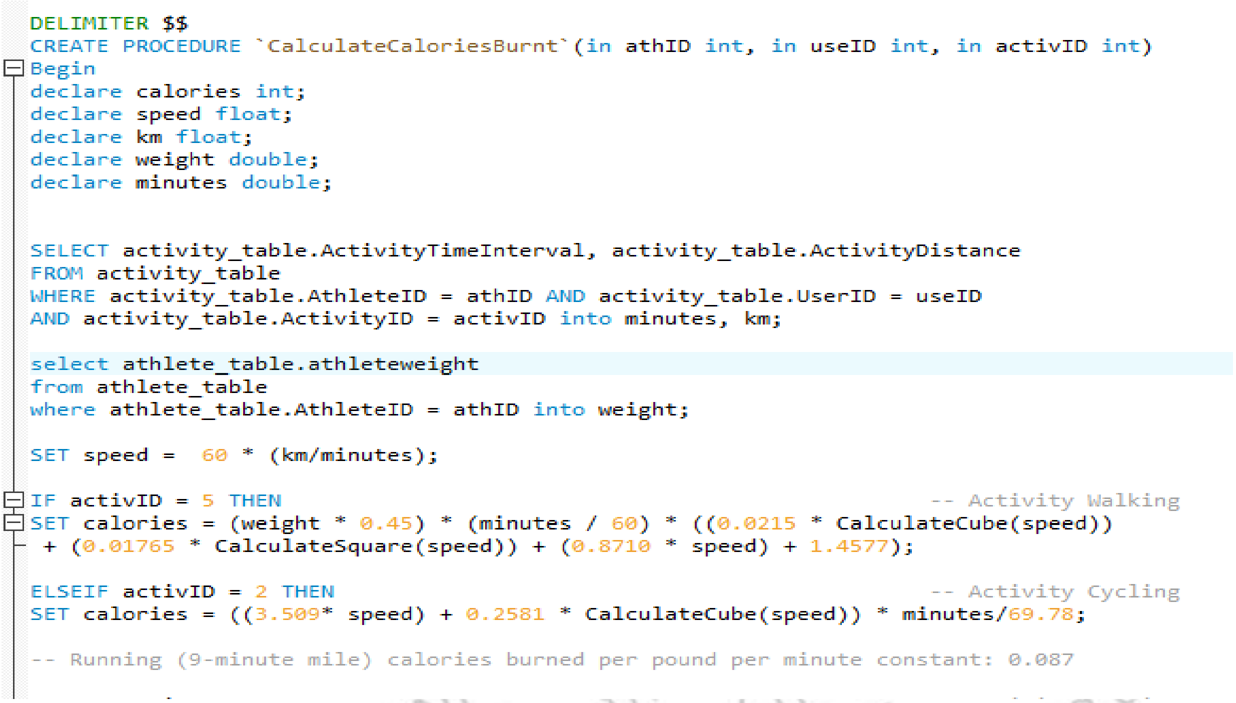
Get Maximum Safe Heart Rate for given age group



```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `GetMaxHR`(in age int, out returnMaxHR int)
BEGIN
DECLARE done INT DEFAULT FALSE;
Declare MaxHR int;
Declare MinAge int;
Declare MaxAge int;

Declare  c_iterator CURSOR for
      SELECT ShrMaxHR, ShrAgeMin, ShrAgeMax from safeheartrate_table;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
OPEN c_iterator;

read_loop: LOOP
 FETCH c_iterator INTO MaxHR, MinAge, MaxAge;

 IF done THEN
      LEAVE read_loop;
 END IF;

 IF age >= MinAge AND age <= MaxAge THEN
    SET returnMaxHR = MaxHR;
  END IF;
END LOOP;
CLOSE c_iterator;
END$$
DELIMITER ;
```

Screenshot 10Get Maximum HR

Get Calories burned for an Athlete in given Activity

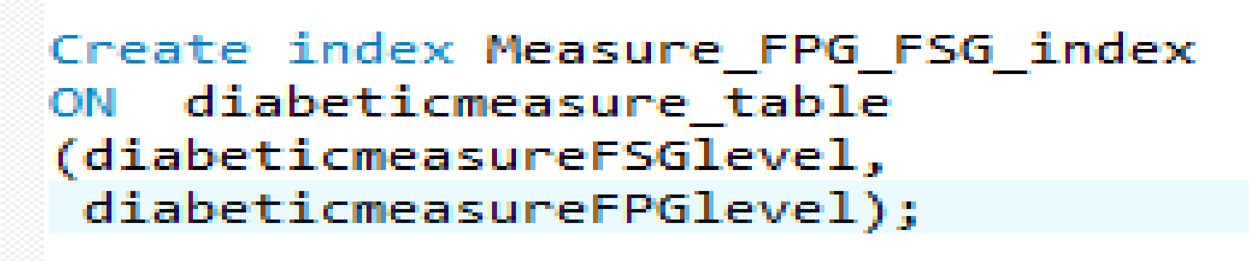


```
DELIMITER $$
CREATE PROCEDURE `CalculateCaloriesBurnt`(in athID int, in useID int, in activID int)
Begin
declare calories int;
declare speed float;
declare km float;
declare weight double;
declare minutes double;


SELECT activity_table.ActivityTimeInterval, activity_table.ActivityDistance
FROM activity_table
WHERE activity_table.AthleteID = athID AND activity_table.UserID = useID
AND activity_table.ActivityID = activID into minutes, km;

select athlete_table.athleteweight
from athlete_table
where athlete_table.AthleteID = athID into weight;

SET speed =  60 * (km/minutes);

IF activID = 5 THEN                                                   -- Activity Walking
SET calories = (weight * 0.45) * (minutes / 60) * ((0.0215 * CalculateCube(speed))
 + (0.01765 * CalculateSquare(speed)) + (0.8710 * speed) + 1.4577);

ELSEIF activID = 2 THEN                                               -- Activity Cycling
SET calories = ((3.509* speed) + 0.2581 * CalculateCube(speed)) * minutes/69.78;

-- Running (9-minute mile) calories burned per pound per minute constant: 0.087
```

Screenshot 11Calculate Calories Burnt

11. INDEX

An index is used to speed up the performance of queries. It does this by reducing the number of database data pages that have to be visited/scanned. In MySQL InnoDB, a clustered index determines the physical order of data in a table. There can be only one clustered index per table (the clustered index IS the table)

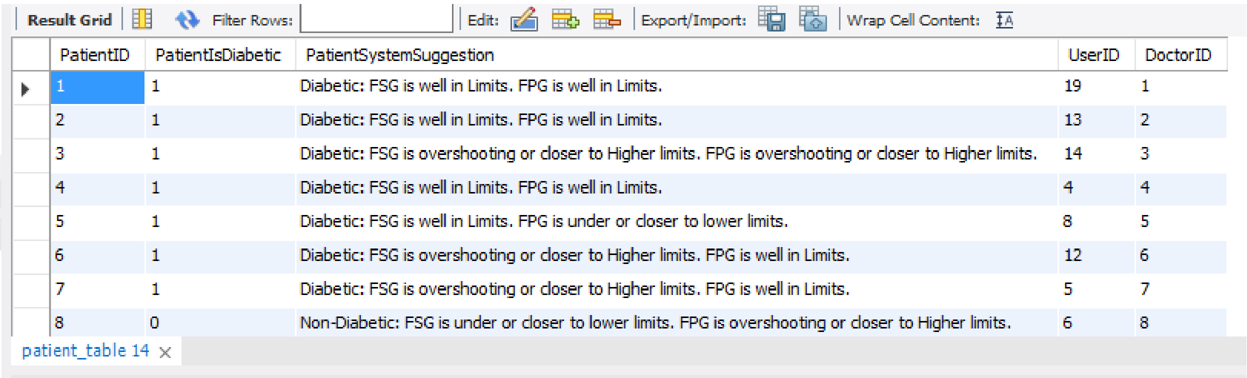


```
Create index Measure_FPG_FSG_index
ON  diabeticmeasure_table
(diabeticmeasureFSGlevel,
 diabeticmeasureFPGlevel);
```

Screenshot 12Index on FPG FSG

12. RESULTS:

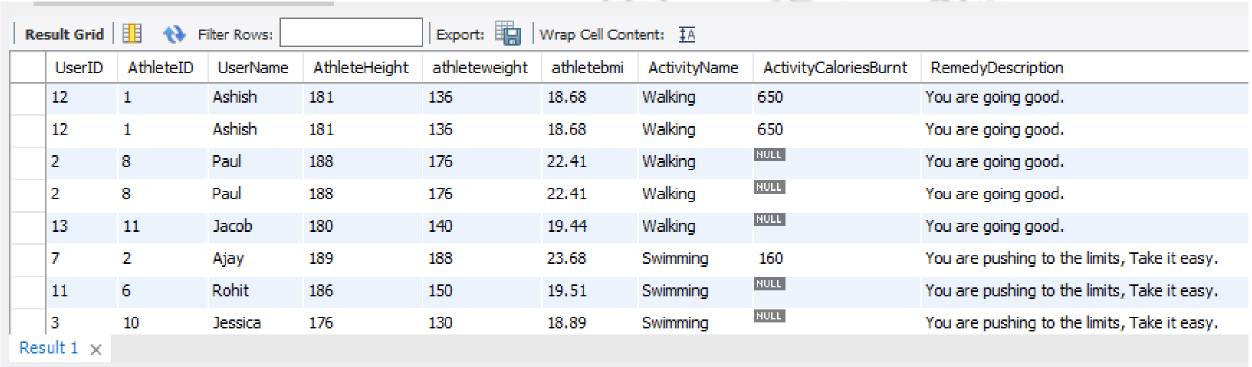
12.1. For Diabetic patient:



PatientID	PatientIsDiabetic	PatientSystemSuggestion	UserID	DoctorID
1	1	Diabetic: FSG is well in Limits. FPG is well in Limits.	19	1
2	1	Diabetic: FSG is well in Limits. FPG is well in Limits.	13	2
3	1	Diabetic: FSG is overshooting or closer to Higher limits. FPG is overshooting or closer to Higher limits.	14	3
4	1	Diabetic: FSG is well in Limits. FPG is well in Limits.	4	4
5	1	Diabetic: FSG is well in Limits. FPG is under or closer to lower limits.	8	5
6	1	Diabetic: FSG is overshooting or closer to Higher limits. FPG is well in Limits.	12	6
7	1	Diabetic: FSG is overshooting or closer to Higher limits. FPG is well in Limits.	5	7
8	0	Non-Diabetic: FSG is under or closer to lower limits. FPG is overshooting or closer to Higher limits.	6	8

Screenshot 13 Result Diabetic Patient

12.2. For Athlete:



UserID	AthleteID	UserName	AthleteHeight	athleteweight	athletebmi	ActivityName	ActivityCaloriesBurnt	RemedyDescription
12	1	Ashish	181	136	18.68	Walking	650	You are going good.
12	1	Ashish	181	136	18.68	Walking	650	You are going good.
2	8	Paul	188	176	22.41	Walking	NULL	You are going good.
2	8	Paul	188	176	22.41	Walking	NULL	You are going good.
13	11	Jacob	180	140	19.44	Walking	NULL	You are going good.
7	2	Ajay	189	188	23.68	Swimming	160	You are pushing to the limits, Take it easy.
11	6	Rohit	186	150	19.51	Swimming	NULL	You are pushing to the limits, Take it easy.
3	10	Jessica	176	130	18.89	Swimming	NULL	You are pushing to the limits, Take it easy.

Screenshot 14 Result Athlete

13. CONCLUSION

- Using Triggers,
- Functions,
- Procedures
- Applying Normalization and
- De-normalization techniques
- Indexing we can retrieve data fast and maintain data easily

14. FUTURE SCOPE

- Can be modified to add animals
- System can be altered to cover other non-invasive techniques
- Can include more activities like horse-riding , skipping, spinning, dancing etc.
- Logging meal intake
- Calorie intake record

15. REFERENCES:

- Noninvasive Method for Glucose Level Estimation by Saliva

  Agrawal RP1*, Sharma N2, Rathore MS2, Gupta VB2, Jain S1, Agarwal V1 and Goyal S1

- 2014 diabetes statistics report-CDC
- Diabetes.org journal
- Walking Calorie Burn Formula
- glucose levels in normal people and in diabetic patients

  D. K. Sen And G. S. Sarin