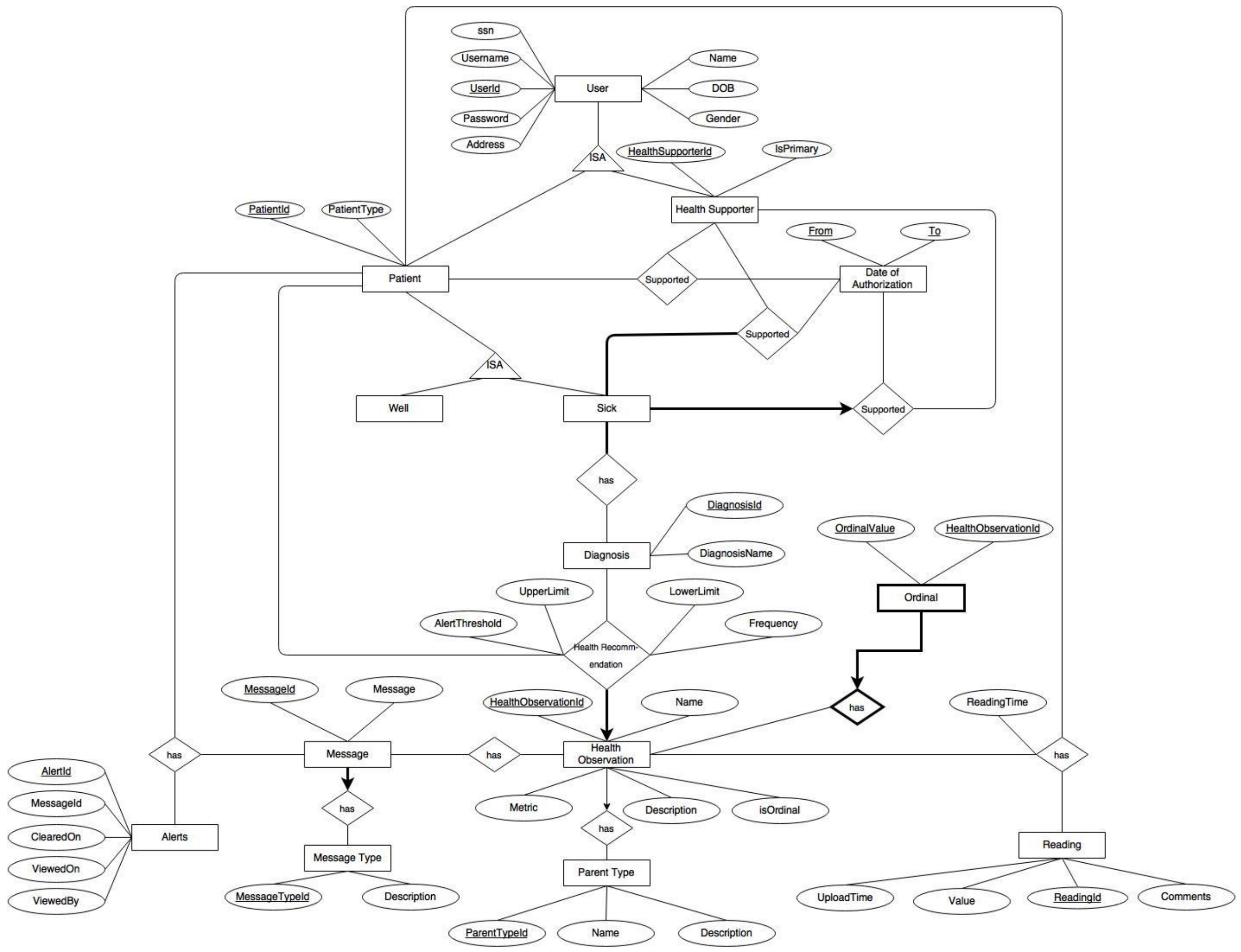
Team details

Anshul chandra	– achand13
Anshuman goel	– agoel5
Bhavesh kasliwal	– bkasliw
Kaustubh gondhalekar	– kgondha

E-R diagram



Entities – data types and functional dependencies

- User

  This table contains basic attributes which are common for all users in the system like name, dob, gender, etc. Userid act as a primary key over here and associated with username.

Attribute	**Data type**	**Key constraints**
Userid	Integer	Primary key
Name	String	not null
Dob	Datetime	
Gender	String	
Address	String	
Username	String	Unique not null
Password	String	not null
Ssn	String	

Functional dependencies

- Userid → { name, dob, gender, address, username, password, ssn }

A primary key constraint is added at userid differentiating each tuple uniquely. Username is also added for simplicity of users which is constrained to be mandatory and also unique throughout the whole database. Several other not null constraints have been added.

Create table users (userid number(22,0) not null, name varchar2(50) not null, dob date, gender varchar2(10), address varchar2(50), username varchar2(50) not null, password varchar2(50) not null, ssn varchar2(50), primary key (userid));

- Patients

  List of userid which are defined as patients in the system.

Attribute	Data type	Key constraints
Patientid	Integer	Primary key
Userid	Integer	Foreign key (user table) not null unique
Patienttype	Enum {well, sick}	not null check

Functional dependencies

  - Patientid → { patientid, userid, patienttype }

Patientid is maintained as primary key for each unique user and maintaining and whether the patient is well or sick.

Create table patients (patientid number(22,0) not null, userid number(22,0) not null, patienttype varchar2(10) not null, primary key (patientid), constraint fkpatientsusers foreign key (userid) references users (userid) on update cascade on delete restrict);

- Health supporter

  List of userid which are defined as health supporters in the system.

Attribute	Data type	Key constraints
Healthsupporterid	Integer	Primary key
Contact	Varchar2	
Userid	Integer	Foreign key (user)

Functional dependencies

  - Healthsupporterid → { contact, userid }

A healthsupporterid is maintained as primary key for all users who are designated as ealth-supporter.

Create table healthsupporter (healthsupporterid number(22,0) not null, userid number(22,0) not null, contact varchar2(20), primary key (healthsupporterid), constraint fkhealthsupporterusers foreign key (userid) references users (userid) on update cascade on delete restrict);

- Patientdiagnosis

  This relationship is maintaining disease that is diagnosed for each patient.

Attribute	**Data type**	**Key constraints**
Patientid	Integer	Composite key
Diagnosisid	Integer	Composite key
Diagnosedon	Timestamp	
Diagnosedtill	Timestamp	
Isactive	Char	

Functional dependencies

- Patientid, diagnosisid → { diagnosedon, diagnosedtill , isactive }

Patientid and diagnosisid are acting as primary key in this table.

Create table patientdiagnosis (patientid number(22,0) not null, diagnosisid number(22,0) not null, diagnosedon timestamp(6), diagnosedtill timestamp(6), isactive char(1), primary key (patientid, diagnosisid), constraint sys_c00247065 foreign key (diagnosisid) references diagnosis (diagnosisid) on update cascade on delete restrict, constraint sys_c00247064 foreign key (patientid) references patients (patientid) on update cascade on delete restrict);

- Diagnosis

  This table lists the diseases that are recorded in the system. Currently, HIV, Heart disease and COPD.

Attribute	**Data type**	**Key constraints**
Diagnosisid	Integer	Primary key
Name	String	

Functional dependencies

- Diagnosisid → { name }

Diagnosisid is acting as primary key of this table.

Create table diagnosis (diagnosisid number(22,0) not null, diagnosisname varchar2(50), primary key (diagnosisid));

- Health observation

  Health observation table contains the different type of observations that are taken into the account.

Attribute	**Data type**	**Key constraints**
Healthobservationid	Integer	Primary key
Healthobservationname	String	
Parenttypeid	Integer	foreign key (parent_type)
Metric	String	
Description	String	
Isordinal	Boolean	

Functional dependencies

- Healthobservationid → { name, parenttypeid, metric, description, isordinal }

Healthobservationid is the primary key for this table with parenttypeid as foreign key.

Create table healthobservations (healthobservationid number(22,0) not null, healthobservationname varchar2(100), parenttypeid number(22,0), metric varchar2(20), description varchar2(100), isordinal char(1), primary key (healthobservationid), constraint sys_c00241631 foreign key (parenttypeid) references parenttype (parenttypeid) on update cascade on delete restrict);

- Health recommendations

  Health recommendation table consists of all recommendations that are added by health supporter for a patient or the default recommendations.

Attribute	**Data type**	**Key constraints**
Healthrecommendationid	Integer	Primary key
Healthobservationid	Integer	Foreign key (health observation)
Diagnosisid	Integer	foreign key (diagnosis)
Patientid	Integer	foreign key (patients)
Alertthreshold	Integer	
Upperlimit	Integer	
Lowerlimit	Integer	
Frequency	Integer	
Inactivitythreshold	Integer	
Consecutivereadings	Integer	

Functional dependencies

- Healthrecommendationid, healthobservationid, diagnosisid, patientid → {alertthreshold, upperlimit, lowerlimit, frequency, inactivitythreshold, consecutivereadings }

Healthrecommendationid s the priary key for this table as each recommendation will be unique and will be and not be based on diagnosisid and patientid i.e., disease specific recommendation, general recommendations and supporter added patient specific recommendations.

Create table healthrecommendations (healthrecommendationid number(22,0) not null, healthobservationid number(22,0) not null, diagnosisid number(22,0) not null, patientid number(22,0), alertthreshold number(22,0), upperlimit number(22,0), lowerlimit number(22,0), frequency number(22,0), inactivitythreshold number(22,0), consecutivereadings number(22,0), primary key (healthrecommendationid), constraint sys_c00268516 foreign key (diagnosisid) references diagnosis (diagnosisid) on update cascade on delete restrict, constraint sys_c00268515 foreign key (healthobservationid) references healthobservations (healthobservationid) on update cascade on delete restrict, constraint sys_c00268517 foreign key (patientid) references patients (patientid) on update cascade on delete restrict);

- Parent_type

  To store what are the different observations that can be recorded.

Attribute	**Data type**	**Key constraints**
Parenttypeid	Integer	Primary key
Name	String	
Description	String	

Functional dependencies

  - Parenttypeid → { name, description }

As each type of observation is unique therefore parenttypeid is acting as primary key for here.

Create table parenttype (parenttypeid number(22,0) not null, name varchar2(50), description varchar2(100), primary key (parenttypeid));

- Ordinal

  Storing the ordinal values.

Attribute	**Data type**	**Key constraints**
Healthobservationid	Integer	Composite key
Ordinalvalue	String	Composite key
Description	String	

Functional dependencies

  - Healthobservationid, ordinalvalue → { description }

Healthobservationid and ordinal value together acting as primary key for recording ordinal type of values.

Create table ordinal (healthobservationid number(22,0) not null, ordinalvalue number(22,0) not null, description varchar2(100), primary key (healthobservationid, ordinalvalue));

- Readings

  This table records all the observations added by the patient or health supporter in the system.

Attribute	Data type	Key constraints
Readingid	Integer	Primary key
Uploadtime	timestamp	
Readingvalue	Decimal	
healthobservationid	Integer	
patientid	Integer	
Observationtime	timestamp	
Comments	String	

Functional dependencies

- Readingid → { uploadtime, readingvalue, healthobservationid, patientid, ordinalvalue , comments }

Readingid is the primary key for every reading that the patient or health supporter adds.

Create table readings (readingid number(22,0) not null, uploadtime timestamp(6), readingvalue number(12,6), healthobservationid number(22,0), comments varchar2(100), patientid number(22,0), observationtime timestamp(6), primary key (readingid), constraint sys_c00241633 foreign key (healthobservationid) references healthobservations (healthobservationid) on update cascade on delete restrict, constraint foreignkey foreign key (patientid) references patients (patientid) on update cascade on delete restrict);

- Alerts

  This table records what type of alerts i.e., low activity and outside the limit alert and when the alert was cleared on.

Attribute	**Data type**	**Key constraints**
Alertid	Integer	Primary key
Alerttypeid	Integer	Foreign key (alerttype)
Clearedon	Datetime	
Timestamp	Datetime	
Clearedtimestamp	Datetime	
Description	String	
Viewedon	Datetime	
Viewedby	Integer	Foreign key (user)

Functional dependencies

- Alertid → { type, clearedon, timestamp, clearedtimestamp, description, viewedon, viewedby }

A unique alerted is maintained for every alert that is generated.

Create table alerts (alertid number(22,0) not null, alerttypeid number(22,0), clearedon timestamp(6), timestamp timestamp(6), viewedon timestamp(6), viewedby number(22,0), readingid number(22,0), patientid number(22,0), description varchar2(400), clearedby number(22,0), primary key (alertid), constraint sys_c00241640 foreign key (viewedby) references users (userid) on update cascade on delete restrict);

- AlertType

  Alert type contains what type of alerts that can be recorded into the system.

Attribute	**Data type**	**Key constraints**
Alerttypeid	Integer	Primary key
Description	String	

Functional dependencies

- Alerttypeid → { description }

Alerttypeid is unique maintaining the different type of alerts that are generated in the system.

Create table alerttype (alerttypeid number(22,0) not null, description varchar2(100), primary key (alerttypeid));

- Hspmap

This relationship is maintaining the relationship of health-supporters that were authorized by the patients and maintaining whether the patient is primary or not and when the health-supporter was authorized.

Attribute	**Data type**	**Key constraints**
Patientid	Integer	Primary key
Healthsupporterid	Integer	Primary key
Isprimary	Char	Not null
Authorizedon	Timestamp	Primary key
Authorizedtill	Timestamp	

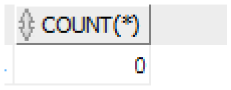
Patientid, healthsupporterid, Authorizedon together acting as a primary key.

Create table hspmap (patientid number(22,0) not null, healthsupporterid number(22,0) not null, isprimary char(1), authorizedon timestamp(6) not null, authorizedtill timestamp(6) not null, primary key (healthsupporterid, patientid, authorizedon));

SQL Queries

1. List the number of health supporters that were authorized in the month of september 2016 by patients suffering from heart disease.

```
Select count(*) from
Hspmap h,patients p,patientdiagnosis pd, diagnosis d
Where
Extract(month from authorizedon)=9 and
Extract(year from authorizedon)=2016 and
P.patientid=h.patientid and
Pd.patientid=p.patientid and
H.patientid=pd.patientid and
D.diagnosisid=pd.diagnosisid and
D.diagnosisname='heart disease';
```



```
COUNT(*)
       0
```

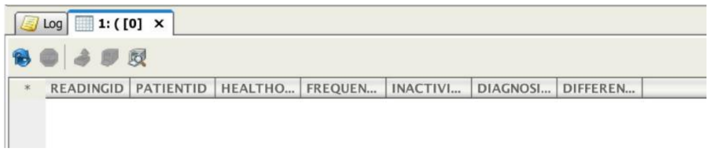
2. Give the number of patients who were not complying with the recommended frequency of recording observations.

```
Select
 *
  from
  (
    select
      rdn.readingid
      ,rdn.patientid
      ,rdn.healthobservationid
      ,nvl(recpatient.frequency, recdefault.frequency) as frequency
      ,nvl(recpatient.inactivitythreshold, recdefault.inactivitythreshold) as
inactivitythreshold
      ,pd.diagnosisid
      ,trunc(current_timestamp) - trunc(rdn.observationtime) as difference
    from
      patientdiagnosis pd
    inner join
      (
        select readingid, readingvalue, patientid, healthobservationid, observationtime
        from
        (
          select readingid, readingvalue, patientid, healthobservationid, observationtime,
rank() over (partition by r.healthobservationid, r.patientid order by r.observationtime
desc) rnk
          from readings r
          --where
            --r.patientid = 6--p_patientid
        ) temp
        where rnk = 1
      ) rdn on(pd.patientid = rdn.patientid)
    left join
      healthrecommendations recpatient on(recpatient.patientid = rdn.patientid and
recpatient.healthobservationid = rdn.healthobservationid and recpatient.diagnosisid =
pd.diagnosisid)
    left join
      healthrecommendations recdefault on(recdefault.patientid is null and
recdefault.healthobservationid = rdn.healthobservationid and recdefault.diagnosisid =
pd.diagnosisid)

    union
    -- checking the recordings against recommendations for well patients
    select
      rdn.readingid
      ,rdn.patientid
      ,rdn.healthobservationid
      ,nvl(recpatient.frequency, recdefault.frequency) as frequency
      ,nvl(recpatient.inactivitythreshold, recdefault.inactivitythreshold) as
inactivitythreshold
      ,0 as diagnosisid
      ,trunc(current_timestamp) - trunc(rdn.observationtime) as difference
    from
      (
        select readingid, readingvalue, patientid, healthobservationid, observationtime
        from
        (
          select readingid, readingvalue, patientid, healthobservationid, observationtime,
rank() over (partition by r.healthobservationid, r.patientid order by r.observationtime
desc) rnk
          from readings r
          --where
            --r.patientid = 6--p_patientid
        ) temp
        where rnk = 1
      ) rdn
    left join
```

```
        healthrecommendations recpatient on(recpatient.patientid = rdn.patientid and
recpatient.healthobservationid = rdn.healthobservationid and recpatient.diagnosisid =
0)
      left join
        healthrecommendations recdefault on(recdefault.patientid is null and
recdefault.healthobservationid = rdn.healthobservationid and recdefault.diagnosisid =
0)
    ) allreading
Where
  difference > (frequency + ((inactivitythreshold/100)*frequency)); select
    *
    from
    (
      select
        rdn.readingid
        ,rdn.patientid
        ,rdn.healthobservationid
        ,nvl(recpatient.frequency, recdefault.frequency) as frequency
        ,nvl(recpatient.inactivitythreshold, recdefault.inactivitythreshold) as
inactivitythreshold
        ,pd.diagnosisid
        ,trunc(current_timestamp) - trunc(rdn.observationtime) as difference
      from
        patientdiagnosis pd
      inner join
        (
          select readingid, readingvalue, patientid, healthobservationid, observationtime
          from
          (
            select readingid, readingvalue, patientid, healthobservationid, observationtime,
rank() over (partition by r.healthobservationid, r.patientid order by r.observationtime
desc) rnk
            from readings r
            --where
              --r.patientid = 6--p_patientid
          ) temp
          where rnk = 1
        ) rdn on(pd.patientid = rdn.patientid)
      left join
        healthrecommendations recpatient on(recpatient.patientid = rdn.patientid and
recpatient.healthobservationid = rdn.healthobservationid and recpatient.diagnosisid =
pd.diagnosisid)
      left join
    healthrecommendations recdefault on(recdefault.patientid is null and
recdefault.healthobservationid = rdn.healthobservationid and recdefault.diagnosisid =
pd.diagnosisid)

   union
   -- checking the recordings against recommendations for well patients
   select
    rdn.readingid
    ,rdn.patientid
    ,rdn.healthobservationid
    ,nvl(recpatient.frequency, recdefault.frequency) as frequency
    ,nvl(recpatient.inactivitythreshold, recdefault.inactivitythreshold) as
inactivitythreshold
    ,0 as diagnosisid
    ,trunc(current_timestamp) - trunc(rdn.observationtime) as difference
   from
    (
     select readingid, readingvalue, patientid, healthobservationid, observationtime
     from
     (
      select readingid, readingvalue, patientid, healthobservationid, observationtime,
rank() over (partition by r.healthobservationid, r.patientid order by r.observationtime
desc) rnk
      from readings r
      --where
       --r.patientid = 6--p_patientid
     ) temp
     where rnk = 1
    ) rdn
   left join
    healthrecommendations recpatient on(recpatient.patientid = rdn.patientid and
recpatient.healthobservationid = rdn.healthobservationid and recpatient.diagnosisid =
0)
   left join
    healthrecommendations recdefault on(recdefault.patientid is null and
recdefault.healthobservationid = rdn.healthobservationid and recdefault.diagnosisid =
0)
  ) allreading
Where
 difference > (frequency + ((inactivitythreshold/100)*frequency));
```



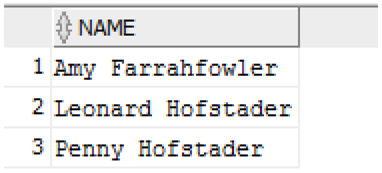
Log | 1: ([0] ×

READINGID	PATIENTID	HEALTHO...	FREQUEN...	INACTIVI...	DIAGNOSI...	DIFFEREN...

3. List the health supporters who themselves are patients.

   Considering patients who are well and sick:

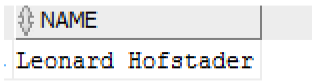
```
Select name from(
Select name,h.userid from healthsupporter h,users u where u.userid=h.userid
Intersect
Select name,p.userid from patients p,users u where u.userid=p.userid);
```



	NAME
1	Amy Farrahfowler
2	Leonard Hofstader
3	Penny Hofstader

   Considering patients who are sick:

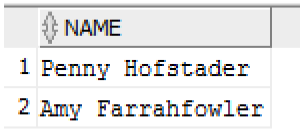
```
select name,h.userid from healthsupporter h,users u where u.userid=h.userid
intersect
select name,p.userid from patients p,users u where u.userid=p.userid and
p.patienttype='Sick');
```



NAME
Leonard Hofstader

4. List the patients who are not ‘sick’.

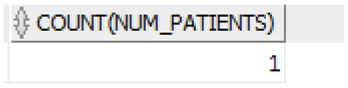
```
Select name from patients p,users u where patienttype<>'sick' and u.userid=p.userid;
```



	NAME
1	Penny Hofstader
2	Amy Farrahfowler

5. How many patients have different observation time and recording time (of the observation).

```
Select count(num_patients) from (select count(*) num_patients from readings where
Extract(year from observationtime)<>extract(year from uploadtime) or
Extract(month from observationtime)<>extract(month from uploadtime) or
Extract(day from observationtime)<>extract(day from uploadtime) or
Extract(hour from observationtime)<>extract(hour from uploadtime) or
Extract(minute from observationtime)<>extract(minute from uploadtime)
Group by patientid);
```



```
COUNT(NUM_PATIENTS)
                  1
```

Constraints

The number of health supporters have been handled using procedures in dbms.
A patient can add disease only if he has health supporter is handled through procedures by invoking them nicely at appropriate position in the application.

Procedures

1.

```
create or replace PROCEDURE AUTHENTICATE
(
  username in users.username%type,
  password in users.password%type,
  isHealthSupporter in number,
  result out number
)
IS
  uId users.userid%type;


BEGIN



  select nvl(u.userid, 0) into uId from  users u where u.username = username and u.password =
password;

  if uId = 0 then
    result := 0;
```

```
   else

     if isHealthSupporter = 1 then
       select nvl(h.healthsupporterid, 0) into result from healthsupporter h where h.userid = uId;
     else
       result := 1;
     end if;

   end if;
   COMMIT;
END;
```

2.

```
create or replace PROCEDURE DeleteHealthSupporter
(
   p_HEALTHSUPPORTERID in HSPMAP.HEALTHSUPPORTERID%TYPE,
   p_PATIENTID in HSPMAP.PATIENTID%TYPE,

   p_output out INTEGER  -- 0: Unable to perform the operation; 1: Health Supporter deleted
successfully;
                         -- 2: Cannot delete since no other health supporter exists
                         -- 3: Invalid healthsupporter / patiend id
)
IS
   isAValidHealthSupporter Integer;
   CountOfOtherHealthSupporters INTEGER;
BEGIN

  p_output := 0;
  CountOfOtherHealthSupporters := 0;

  SELECT count(HealthSupporterId) into isAValidHealthSupporter FROM HSPMAP WHERE
PATIENTID = p_PATIENTID AND HEALTHSUPPORTERID = p_HEALTHSUPPORTERID;
  SELECT count(HealthSupporterId) into CountOfOtherHealthSupporters FROM HSPMAP
WHERE PATIENTID = p_PATIENTID AND HEALTHSUPPORTERID <>
p_HEALTHSUPPORTERID;

  IF(isAValidHealthSupporter > 0) THEN
   IF(CountOfOtherHealthSupporters > 0 AND p_output = 0) THEN
     DELETE FROM HSPMAP WHERE HEALTHSUPPORTERID = p_HEALTHSUPPORTERID
AND PATIENTID = p_PATIENTID;

     -- Make the other health supporter as primary
     UPDATE HSPMAP
       SET ISPRIMARY        = 'Y'
```

```
    WHERE PATIENTID       = p_PATIENTID
      AND HEALTHSUPPORTERID <> p_HEALTHSUPPORTERID
      AND ISPRIMARY = 'N';

    p_output := 1; -- Health supporter deleted successfully
   ELSE
    p_output := 2;  -- Cannot delete since no other health supporter exists
   END IF;

  ELSE
   p_output := 3; -- Invalid healthSupporterId / patientId
  END IF;

  COMMIT;
END;
```

3.

```
create or replace PROCEDURE GenerateOverTheLimitAlerts (
  p_PatientId in Patients.PatientId%Type,

  p_output out int
)
IS
BEGIN

  p_output := 0;

  -- Generate the alerts for the user and store it in alerts table
  INSERT INTO ALERTS
  (
   ALERTID,
   ALERTTYPEID,
   CLEAREDON,
   TIMESTAMP,
   ClearedBy,
   Description,
   VIEWEDON,
   VIEWEDBY,
   READINGID,
   PATIENTID
  )
  SELECT
   (SELECT (NVL(MAX(ALERTID), 0) + 1) FROM ALERTS)
   ,1
   ,NULL
   ,CURRENT_TIMESTAMP
   ,NULL
   ,Alertmessage
   ,NULL
   ,NULL
   ,MaxReadingId
   ,p_PatientId
  FROM
  (
   SELECT
     DiagnosisId, HealthObservationId, HealthobservationName, MaxReadingId,
ConsecutiveReadings, AlertThreshold, Count(OverLimit) countOfOverlimits
     ,Count(OverLimit) || ' out of last ' || ConsecutiveReadings || ' readings for ' ||
HealthobservationName || ' are outside the recommended limits' as AlertMessage
   FROM
   (
     SELECT
       rdn.readingid
       ,rdnMax.readingid as MaxReadingId
       ,rdn.patientId
       ,rdn.healthobservationid
       ,NVL(recPatient.ConsecutiveReadings, recDefault.ConsecutiveReadings) as
ConsecutiveReadings
       ,NVL(recPatient.AlertThreshold, recDefault.AlertThreshold) as AlertThreshold
       ,rdn.observationTime
       ,rdn.readingValue
       ,pd.DiagnosisId
       ,NVL(recPatient.LowerLimit, recDefault.LowerLimit) as LowerLimit
       ,NVL(recPatient.upperLimit, recDefault.UpperLimit) as UpperLimit
       ,ob.HealthObservationName
       ,CASE WHEN
           (rdn.readingValue >= NVL(recPatient.LowerLimit, recDefault.LowerLimit) and
rdn.readingValue <= NVL(recPatient.upperLimit, recDefault.UpperLimit))
            OR (NVL(recPatient.LowerLimit, recDefault.LowerLimit) IS NULL and
NVL(recPatient.upperLimit, recDefault.UpperLimit) IS NULL) THEN 0
           ELSE 1 END AS OverLimit
     FROM
      (
        select readingid, readingvalue, patientid, healthobservationid, observationTime
        from
        (
         select readingid, readingvalue, patientid, healthobservationid, observationTime, rank()
over (partition by r.HEALTHOBSERVATIONID order by r.ObservationTime desc) rnk
         from readings r
         where
```

```
          r.patientid = p_PatientId
        ) temp
        where rnk <= (SELECT ConsecutiveReadings FROM HealthRecommendations WHERE
healthObservationID = temp.healthobservationID and patientId = temp.patientid)
      ) rdn
      INNER JOIN
      (
        select readingid, healthobservationid, observationTime
        from
        (
          select readingid, patientid, healthobservationid, observationTime, rank() over (partition
by r.HEALTHOBSERVATIONID order by r.ObservationTime desc) rnk
          from readings r
          where
            r.patientid = p_PatientId
        ) temp
        where rnk = 1
      ) rdnMax ON(rdn.healthobservationid = rdnMax.healthobservationid)
    INNER JOIN
      HealthObservations ob ON(rdn.HealthObservationId = ob.HealthObservationId and
ob.IsOrdinal = 'N')
    INNER JOIN
      PatientDiagnosis pd ON(pd.PatientId = rdn.PatientId)
    LEFT JOIN
      HealthRecommendations recPatient ON(recPatient.PatientId = rdn.PatientId AND
recPatient.HealthObservationId = ob.HEALTHOBSERVATIONID AND recPatient.DiagnosisId
= pd.DiagnosisId)
    LEFT JOIN
      HealthRecommendations recDefault ON(recDefault.PatientId IS NULL AND
recDefault.HealthObservationId = ob.HEALTHOBSERVATIONID AND recDefault.DiagnosisId
= pd.DiagnosisId)
  ) allReading
  WHERE
    OverLimit = 1
  GROUP BY
    DiagnosisId, HealthObservationId, ConsecutiveReadings, AlertThreshold, MaxReadingId,
healthobservationname
  HAVING
    (Count(OverLimit) > (Alertthreshold/100)* ConsecutiveReadings)
 ) tmp
LEFT OUTER JOIN
  Alerts al ON(
          al.PatientId = p_PatientId
          and al.readingId = tmp.MaxReadingId
          and al.alertTypeId = 1)
WHERE
  al.alertId is null;

p_output := 1;

COMMIT;
END;
```

4.

```
create or replace PROCEDURE getHSListForPatient(
            p_PatientId IN HSPMAP.PATIENTID%TYPE,
            c_list OUT SYS_REFCURSOR)
IS
BEGIN

  OPEN c_list FOR

  SELECT
    u.userid
    ,u.name
    ,u.username
    ,hs.CONTACT
    ,h.IsPrimary
    ,to_char(h.AuthorizedOn, 'yyyy-MM-dd') as AuthorizedOn
    ,to_char(h.AuthorizedTill, 'yyyy-MM-dd') as AuthorizedTill
    ,hs.HealthSupporterId
  FROM
    HSPMAP h
  INNER JOIN
    HealthSupporter hs ON(h.HealthSupporterId = hs.HealthSupporterId)
  INNER JOIN
    Users u ON(u.UserId = hs.UserId)
  WHERE
    h.PatientId = p_PatientId
    and rownum <=2

  ORDER BY isPrimary desc;

END;
```

5.

```
create or replace PROCEDURE getPatientListForHS(
            p_HealthSupporterId IN HSPMAP.HEALTHSUPPORTERID%TYPE,
            c_patients OUT SYS_REFCURSOR)
IS
```

```
BEGIN

  OPEN c_patients FOR

  SELECT
    u.USERID,
    u.NAME,
    u.DOB,
    u.GENDER,
    u.ADDRESS,
    u.SSN,
    p.PATIENTID
  FROM
    HSPMAP hs, PATIENTS p, USERS u
  WHERE
    hs.PATIENTID = p.PATIENTID
    AND p.USERID = u.USERID
    AND hs.HEALTHSUPPORTERID = p_HealthSupporterId;

END;
```

6.

```
create or replace PROCEDURE getUserDiseases(
                p_PatientId IN HSPMAP.HEALTHSUPPORTERID%TYPE,
                c_diseases OUT SYS_REFCURSOR)
IS
BEGIN

  OPEN c_diseases FOR
  select d.diagnosisid, d.diagnosisname, p.patientid
  from
    diagnosis d
  left outer join
  (select pd.patientid, pd.diagnosisid from patientdiagnosis pd where pd.patientid = p_PatientId
) p
  on (d.diagnosisid = p.diagnosisid)
  where d.diagnosisid <> 0;
END;
```

7.

```
create or replace PROCEDURE INSERTREADING
(
    r_patientId in readings.patientId%type,
    r_readingValue in readings.readingvalue%type,
    r_hid in readings.healthobservationid%type,
   r_observationtime in readings.observationtime%type,
   r_comments in readings.comments%type

)

IS
  r_rid readings.readingId%type;

BEGIN

  Select NVL(max(readingid), 0) into r_rid from readings;
  r_rid := r_rid + 1;
  insert into readings values (r_rid, current_timestamp, r_readingValue, r_hid, r_comments,
r_patientId, r_observationtime);
  COMMIT;
END;
```

8.

```
create or replace PROCEDURE InsertUpdateHealthSupporter
(
   p_USERID in Users.UserId%Type,
   p_HEALTHSUPPORTERID in HEALTHSUPPORTER.HEALTHSUPPORTERID%TYPE,
   p_PATIENTID in PATIENTS.PATIENTID%TYPE,
   p_ISPRIMARY in HSPMAP.ISPRIMARY%type,
   p_Contact in HEALTHSUPPORTER.CONTACT%TYPE,
   p_AUTHORIZEDON in HSPMAP.AUTHORIZEDON%TYPE,
   p_AUTHORIZEDTILL in HSPMAP.AUTHORIZEDTILL%TYPE,
   p_output out INTEGER  -- 0: Unable to perform the operation;
                     -- 1: Health Supporter tagged successfully;
                     -- 2: Health Supporter tagged as a Primary health supporter;
                     -- 3: Health Supporter details updated successfully;
                     -- 4: No other primary health supporter exists;
                     -- 5: Invalid health supporter id;
                     -- 6: User already tagged as a health supporter
                     -- 7: Cannot add healthsupporter since already two health supporters exist
                     -- 8: Health supporter tagged as a secondary health supporter
)
IS
   isPrimary HSPMAP.ISPRIMARY%type;
   newHealthSupporterId HSPMAP.HealthSupporterId%TYPE;
   counterHealthSupporter INTEGER;
   counterPrimaryHealthSupporter INTEGER;
BEGIN

   p_output := 0;
```

```
  isPrimary := p_ISPRIMARY;
  newHealthSupporterId := 0;

  IF(p_HEALTHSUPPORTERID = 0) THEN

    SELECT count(healthSupporterId) INTO counterHealthSupporter
    FROM
      HSPMAP
    WHERE
      PATIENTID = p_PATIENTID;
      --AND ((to_timestamp(p_AUTHORIZEDON) between AUTHORIZEDON and
AUTHORIZEDTILL) or (to_timestamp(p_AUTHORIZEDTILL) between AUTHORIZEDON and
AUTHORIZEDTILL));

    IF(counterHealthSupporter < 2) THEN
      -- We can add since no. of health supporters is less than 2
      SELECT count(HealthSupporterId) into newHealthSupporterId from HealthSupporter
WHERE USERID = p_USERID;

      -- Check if the user is already registered
      IF(newHealthSupporterId = 0) THEN
        Select NVL(max(HealthSupporterId), 0) into newHealthSupporterId from
HealthSupporter;
        newHealthSupporterId := newHealthSupporterId + 1;

        INSERT INTO HEALTHSUPPORTER (HEALTHSUPPORTERID, USERID, CONTACT)
        VALUES (newHealthSupporterId, p_USERID, p_Contact);
      ELSE
         SELECT HealthSupporterId into newHealthSupporterId from HealthSupporter WHERE
USERID = p_USERID;
      END IF;

      SELECT
        COUNT(HealthSupporterId) into counterPrimaryHealthSupporter
      FROM
        HSPMAP
      WHERE
        PATIENTID = p_PATIENTID
        AND ISPRIMARY = 'Y'
        AND HEALTHSUPPORTERID <> newHealthSupporterId;
        --AND ((to_timestamp(p_AUTHORIZEDON) between AUTHORIZEDON and
AUTHORIZEDTILL) or (to_timestamp(p_AUTHORIZEDTILL) between AUTHORIZEDON and
AUTHORIZEDTILL));

      IF(p_ISPRIMARY = 'N') THEN
```

```
 IF(counterPrimaryHealthSupporter = 0) THEN
  -- Make the current health supporter as no other primary health supporter exists
  isPrimary := 'Y';
  p_output := 10;
 END IF;
ELSE
 IF(counterPrimaryHealthSupporter > 0) THEN
  -- Make the current health supporter as no other primary health supporter exists
  isPrimary := 'N';
  p_output := 11;
 END IF;
END IF;



counterPrimaryHealthSupporter := 0;

SELECT
 COUNT(HealthSupporterId) into counterPrimaryHealthSupporter
FROM
 HSPMAP
WHERE
 PATIENTID = p_PATIENTID
 AND HEALTHSUPPORTERID = newHealthSupporterId;
 --AND ((to_timestamp(p_AUTHORIZEDON) between AUTHORIZEDON and
AUTHORIZEDTILL) or (to_timestamp(p_AUTHORIZEDTILL) between AUTHORIZEDON and
AUTHORIZEDTILL));

IF(counterPrimaryHealthSupporter = 0) THEN
 -- Tagging the Health Supporter to a user
 INSERT INTO HSPMAP
 (
  PATIENTID,
  HEALTHSUPPORTERID,
  ISPRIMARY,
  AUTHORIZEDON,
  AUTHORIZEDTILL
 )
 VALUES
 (
  p_PATIENTID,
  newHealthSupporterId,
  isPrimary,
  p_AUTHORIZEDON,
  to_timestamp('31-12-4712')
```

```
      );

      if(p_output = 10) then
        p_output := 2;
      else if(p_output = 11) then
          p_output := 8;
        else
          p_output := 1;
        end if;
      end if;
    ELSE
      p_output := 6; -- User already tagged as a health supporter
    END IF;

  ELSE
    p_output := 7;  -- Cannot add healthsupporter since already two health supporters exist
  END IF;
 ELSE
  Select COUNT(HealthSupporterId) into counterHealthSupporter FROM HealthSupporter
WHERE HEALTHSUPPORTERID = p_HEALTHSUPPORTERID;

  IF(counterHealthSupporter > 0) THEN

    IF(p_ISPRIMARY = 'N') THEN
      -- Check if another health supporter exists
      SELECT COUNT(HealthSupporterId) into counterPrimaryHealthSupporter
      FROM
        HSPMAP
      WHERE
        PATIENTID = p_PATIENTID AND ISPRIMARY = 'N'
        AND HEALTHSUPPORTERID <> p_HEALTHSUPPORTERID;

      IF(counterPrimaryHealthSupporter > 0) THEN

        UPDATE HSPMAP
        SET ISPRIMARY        = p_ISPRIMARY
          --,AUTHORIZEDON     = p_AUTHORIZEDON
          --,AUTHORIZEDTILL   = p_AUTHORIZEDTILL
        WHERE PATIENTID       = p_PATIENTID
          AND HEALTHSUPPORTERID = p_HEALTHSUPPORTERID;

        -- Make the other health supporter as primary
        UPDATE HSPMAP
          SET ISPRIMARY        = 'Y'
        WHERE PATIENTID       = p_PATIENTID
          AND HEALTHSUPPORTERID <> p_HEALTHSUPPORTERID
          AND ISPRIMARY = 'N';

        -- Updating the contact
        UPDATE HEALTHSUPPORTER
        SET CONTACT = p_Contact
        WHERE
          HEALTHSUPPORTERID = p_HEALTHSUPPORTERID;

        p_output := 3;
      ELSE
        p_output := 4; -- No other health supporter exists
      END IF;

    ELSE
      -- Check if another health supporter exists who is primary
      SELECT COUNT(HealthSupporterId) into counterPrimaryHealthSupporter
      FROM
        HSPMAP
      WHERE
        PATIENTID = p_PATIENTID AND ISPRIMARY = 'Y'
        AND HEALTHSUPPORTERID <> p_HEALTHSUPPORTERID;

      UPDATE HSPMAP
        SET ISPRIMARY        = p_ISPRIMARY
          --,AUTHORIZEDON     = p_AUTHORIZEDON
          --,AUTHORIZEDTILL   = p_AUTHORIZEDTILL
        WHERE PATIENTID       = p_PATIENTID
          AND HEALTHSUPPORTERID = p_HEALTHSUPPORTERID;

      IF(counterPrimaryHealthSupporter > 0) THEN
        -- Make the other health supporter as primary
        UPDATE HSPMAP
          SET ISPRIMARY        = 'N'
        WHERE PATIENTID       = p_PATIENTID
          AND HEALTHSUPPORTERID <> p_HEALTHSUPPORTERID;


      END IF;

      UPDATE HEALTHSUPPORTER
      SET CONTACT = p_Contact
      WHERE
        HEALTHSUPPORTERID = p_HEALTHSUPPORTERID;
```

```
        p_output := 3;
      END IF;

    ELSE
      p_output := 5; -- Invalid Health supporter ID

    END IF;

  END IF;

  COMMIT;
END;
```

9.

```
create or replace PROCEDURE InsertUpdatePatient
(
  p_USERID in Users.UserId%Type,
  p_NAME in Users.NAME%Type,
  p_DOB in USERS.DOB%type,
  p_GENDER in USERS.GENDER%type,
  p_ADDRESS in USERS.ADDRESS%Type,
  p_USERNAME in USERS.USERNAME%type,
  p_PASSWORD in USERS.PASSWORD%type,
  p_SSN in USERS.SSN%type
)
IS
  newUserId Users.UserId%TYPE;
  newPatientId PATIENTS.PATIENTID%Type;
BEGIN

  if p_USERID = 0 then

    Select NVL(max(UserId), 0) into newUserId from USERS;

    newUserId := newUserId + 1;

    INSERT INTO USERS
    (
      USERID,
      NAME,
      DOB,
      GENDER,
      ADDRESS,
      USERNAME,
      PASSWORD,
      SSN
    )
    VALUES
    (
      newUserId,
      p_NAME,
      p_DOB,
      p_GENDER,
      p_ADDRESS,
      p_USERNAME,
      p_PASSWORD,
      p_SSN
    );

    Select NVL(max(PatientId), 0) into newPatientId from PATIENTS;

    newPatientId := newPatientId + 1;

    INSERT INTO PATIENTS
    (PATIENTID, USERID, PATIENTTYPE)
    VALUES(newPatientId, newUserId, 'Well');

  else

    UPDATE USERS
    SET
       NAME     = p_NAME
      ,DOB      = p_DOB
      ,GENDER   = p_GENDER
      ,ADDRESS  = p_ADDRESS
      ,PASSWORD = p_PASSWORD
      ,SSN      = p_SSN
    WHERE
      USERID     = p_USERID;
  end if;

  COMMIT;
END;
```

10.

```
create or replace PROCEDURE InsertUpdateRecommendation
(
  p_HealthRecommendationId in
HEALTHRECOMMENDATIONS.HEALTHRECOMMENDATIONID%type,
  p_HEALTHOBSERVATIONID IN
HEALTHRECOMMENDATIONS.HEALTHOBSERVATIONID%Type,
  p_DIAGNOSISID IN HEALTHRECOMMENDATIONS.DIAGNOSISID%type,
  p_PATIENTID IN HEALTHRECOMMENDATIONS.PATIENTID%Type,

  p_AlertThreshold in HEALTHRECOMMENDATIONS.ALERTTHRESHOLD%type,
  p_UpperLimit in HEALTHRECOMMENDATIONS.UPPERLIMIT%type, -- This can be
normal value or ordinal value depending upon the diagnosis
  p_LowerLimit in HEALTHRECOMMENDATIONS.LOWERLIMIT%type, -- This can be
normal value or ordinal value depending upon the diagnosis
  p_Frequency in HEALTHRECOMMENDATIONS.FREQUENCY%type,

  p_InactivityThreshold in HEALTHRECOMMENDATIONS.INACTIVITYTHRESHOLD%type,
  p_ConsecutiveReadings in
HEALTHRECOMMENDATIONS.CONSECUTIVEREADINGS%type,

  p_output out INTEGER  -- 0: Unable to process
                  -- 1: Health recommendation added successfully
                  -- 2: A default health recommendation already exists for the health observation
                  -- 3: Updated the existing health recommendation for the patient
                  -- 4: Health recommendation updated successfully
                  -- 5: Switching back to default configuration
)
IS
  checkExistingRecord INTEGER;
  checkDefaultValue integer;
  newHealthRecommendationId
HEALTHRECOMMENDATIONS.HEALTHRECOMMENDATIONID%type;
BEGIN

  p_output := 0;
  checkExistingRecord := 0;
  newHealthRecommendationId := 0;

  IF(p_HealthRecommendationId = 0) THEN
  -- Case : Add new recommendation

  -- Check for existing records
  SELECT
    COUNT(HEALTHRECOMMENDATIONID) INTO checkExistingRecord
  FROM
    HEALTHRECOMMENDATIONS
  WHERE
    HEALTHOBSERVATIONID   = p_HEALTHOBSERVATIONID
    AND DIAGNOSISID       = p_DIAGNOSISID
```

```
    AND PATIENTID        = p_PATIENTID;
    --AND ALERTTHRESHOLD    = p_ALERTTHRESHOLD
    --AND UPPERLIMIT       = p_UPPERLIMIT
    --AND LOWERLIMIT       = p_LOWERLIMIT
    --AND FREQUENCY        = p_FREQUENCY;

   IF(checkExistingRecord = 0) THEN

    SELECT
     COUNT(HEALTHRECOMMENDATIONID) INTO checkDefaultValue
    FROM
     HEALTHRECOMMENDATIONS
    WHERE
     HEALTHOBSERVATIONID     = p_HEALTHOBSERVATIONID
     AND DIAGNOSISID        = p_DIAGNOSISID
     AND ALERTTHRESHOLD      = p_ALERTTHRESHOLD
     AND UPPERLIMIT         = p_UPPERLIMIT
     AND LOWERLIMIT         = p_LOWERLIMIT
     AND FREQUENCY          = p_FREQUENCY
     AND INACTIVITYTHRESHOLD = p_InactivityThreshold
     AND CONSECUTIVEREADINGS = p_ConsecutiveReadings
     AND PATIENTID IS NULL;


    IF(checkDefaultValue = 0) THEN
     -- Add the health recommendation
     Select NVL(max(HEALTHRECOMMENDATIONID), 0) into newHealthRecommendationId
from HEALTHRECOMMENDATIONS;

     newHealthRecommendationId := newHealthRecommendationId + 1;

     INSERT INTO HEALTHRECOMMENDATIONS
     (
      HEALTHRECOMMENDATIONID,
      HEALTHOBSERVATIONID,
      DIAGNOSISID,
      PATIENTID,
      ALERTTHRESHOLD,
      UPPERLIMIT,
      LOWERLIMIT,
      FREQUENCY,
      INACTIVITYTHRESHOLD,
      CONSECUTIVEREADINGS
     )
     VALUES
```

```
      (
       newHealthRecommendationId,
       p_HEALTHOBSERVATIONID,
       p_DIAGNOSISID,
       p_PATIENTID,
       p_ALERTTHRESHOLD,
       p_UPPERLIMIT,
       p_LOWERLIMIT,
       p_FREQUENCY,
       p_InactivityThreshold,
       p_ConsecutiveReadings
      );

      p_output := 1; -- Health recommendation added successfully
     ELSE
      p_output := 2; -- A deafault health recommendation already exists for the health
observation
     END IF;
     --end Default check
    ELSE
     -- Updating the record
     UPDATE HEALTHRECOMMENDATIONS
     SET HEALTHOBSERVATIONID     = p_HEALTHOBSERVATIONID
      ,DIAGNOSISID             = p_DIAGNOSISID
      ,ALERTTHRESHOLD          = p_ALERTTHRESHOLD
      ,UPPERLIMIT              = p_UPPERLIMIT
      ,LOWERLIMIT              = p_LOWERLIMIT
      ,FREQUENCY               = p_FREQUENCY
      ,INACTIVITYTHRESHOLD     = p_InactivityThreshold
      ,CONSECUTIVEREADINGS     = p_ConsecutiveReadings
     WHERE
      HEALTHRECOMMENDATIONID = (SELECT
                                 HEALTHRECOMMENDATIONID
                                FROM
                                 HEALTHRECOMMENDATIONS
                                WHERE
                                 HEALTHOBSERVATIONID   = p_HEALTHOBSERVATIONID
                                 AND DIAGNOSISID       = p_DIAGNOSISID
                                 AND PATIENTID         = p_PATIENTID
                                );

     p_output := 3; -- Updated the existing health recommendation of the person
    END IF;
    -- end uniqueness check
   ELSE
```

```
    -- Update the health recommendation

    -- check if an entry with default values same as the updated record exist
    SELECT
      COUNT(HEALTHRECOMMENDATIONID) INTO checkDefaultValue
    FROM
      HEALTHRECOMMENDATIONS
    WHERE
      HEALTHOBSERVATIONID     = p_HEALTHOBSERVATIONID
      AND DIAGNOSISID         = p_DIAGNOSISID
      AND ALERTTHRESHOLD      = p_ALERTTHRESHOLD
      AND UPPERLIMIT          = p_UPPERLIMIT
      AND LOWERLIMIT          = p_LOWERLIMIT
      AND FREQUENCY           = p_FREQUENCY
      AND INACTIVITYTHRESHOLD = p_InactivityThreshold
      AND CONSECUTIVEREADINGS = p_ConsecutiveReadings
      AND PATIENTID IS NULL;

     IF(checkDefaultValue = 0) THEN
      -- Updating the record
      UPDATE HEALTHRECOMMENDATIONS
      SET HEALTHOBSERVATIONID     = p_HEALTHOBSERVATIONID
        ,DIAGNOSISID            = p_DIAGNOSISID
        ,ALERTTHRESHOLD         = p_ALERTTHRESHOLD
        ,UPPERLIMIT             = p_UPPERLIMIT
        ,LOWERLIMIT             = p_LOWERLIMIT
        ,FREQUENCY              = p_FREQUENCY
        ,INACTIVITYTHRESHOLD     = p_InactivityThreshold
        ,CONSECUTIVEREADINGS     = p_ConsecutiveReadings
      WHERE
        HEALTHRECOMMENDATIONID = p_HEALTHRECOMMENDATIONID;

      p_output := 4; -- Health recommendation updated successfully

     ELSE
      -- Delete the existing record since a default value already exists
      DELETE FROM HEALTHRECOMMENDATIONS WHERE
HEALTHRECOMMENDATIONID = p_HEALTHRECOMMENDATIONID;

      p_output := 4; -- Health recommendation updated successfully
     END IF;
     -- end default check
  END IF;
  -- end Update
END;
```

Inactivity alert Query for insertion:

```
insert into ALERTS
select alert_id_final, alert_type,null,current_timestamp,null,null,readingid,?,
concat('Inactivity to record observations for ',HEALTHOBSERVATIONNAME),null
from
(
select alert_id+rn as alert_id_final, alert_id_g.* from
(
select final.*, ROW_NUMBER() OVER (PARTITION BY alert_id order by readingid) AS rn
from
(
select (Select NVL(max(alertid), 0) from alerts)as alert_id,
2 as alert_type,readingid,
HEALTHOBSERVATIONNAME
from
(
select * from (


(
  select r2.healthobservationid,max(r2.readingid) as readingid from readings r2 where
(r2.HEALTHOBSERVATIONID,r2.PATIENTID) not in(
  select r.HEALTHOBSERVATIONID,r.patientid from readings r,
  (
  select * from (
  select a.*, ROW_NUMBER()
    OVER (PARTITION BY a.HEALTHOBSERVATIONID ORDER BY a.frequency) AS freq
from
  (
  select * from healthrecommendations where patientid=? and diagnosisid in (select
diagnosisid from patientdiagnosis where patientid=? and isactive='Y')
  )a
  )b
  where b.freq= 1
union
   select healthrecommendations.*, 0 as freq from healthrecommendations where patientid
is null and diagnosisid in (select diagnosisid from patientdiagnosis where patientid=? and
isactive='Y')
   and HEALTHOBSERVATIONID not in (select HEALTHOBSERVATIONID from
healthrecommendations where patientid=? and diagnosisid in (select diagnosisid from
patientdiagnosis where patientid=? and isactive='Y')
    ) ) temp1
 where r.HEALTHOBSERVATIONID=temp1.HEALTHOBSERVATIONID
 and r.patientid=temp1.patientid
 and temp1.patientid=?
 and current_timestamp-
(temp1.frequency+temp1.frequency*.01*temp1.inactivitythreshold)<r.OBSERVATIONTIME
) and r2.PATIENTID=?
 group by r2.healthobservationid)a
 inner join
 HEALTHOBSERVATIONs ho on ho.HEALTHOBSERVATIONID =
a.HEALTHOBSERVATIONID
 )
)
)final
)alert_id_g
)insert_alert
 ;
```

Application constraints

- Well patients should not be tagged to any disease at a given time.
- Sick patients should be tagged to at least one disease (diagnosis) at a given time.
- Health supporters are designated by patients
- A patient can have a maximum of 2 health supporters – primary and secondary.
- A sick patient should have at least 1 health supporter
- Prior to authorization date, health supporters should not have access to patient information.
- For a disease class, patients directly inherit the observation recommendations for that class of patients (unless a specific recommendation is made)
- For well patient observation requirements are merely recommendations. For sick patients, observation requirements are mandatory.
- Some observation types may be inapplicable to particular groups of patients but applicable for others.
- Each sick patient class may include recommendations that override those for well patients.
- Two types of alerts are available - outside-the-limit, low-activity-alert.
  - Outside-the-limit alert when a certain threshold percentage number of consecutive readings are over the specified limits for the patient. Outside-the-limit thresholds are specific for each patient.
  - Low-activity-alerts which help to identify patients that seem to be disengaged from the system. If the recommended frequency of an observation type is x and patients haven't recorded any activity by certain threshold beyond x.
- Alerts can be cleared in one of two ways: either the health supporter clears them (essentially representing the fact that they have intervened in some way) or a patient enters an observation for the missing observation type.
- A user shouldn't just be able to clear alerts before seeing them.
- For entering observation data, only available options should be the options for observation types associated with the patient classes that the patient belongs to.