```
/*
==============================================================================================
Summary:
This file consists of all SQL queries for creating tables, procedures, functions and triggers.
The procedures, functions and triggers are written to enforce constraints that are not
defined in the table definition.

Each SQL query below has the name and description of the procedures/functions/triggers.
==============================================================================================
*/



/*
=================================================================================================
====================================
RELATIONAL MODEL
=================================================================================================
====================================
*/

/*
Name: people
Description: Table consists of login credentials and details of a person registering as a
patient or a health supporter. If a person is registering as a patient then the patient_flag is
set to 1. If a person is registering as a health_supporter then the health_supporter_flag is
set to 1.
Functional Dependencies: pid functionally determines all other columns.
                         pid -> first_name, last_name, date_of_birth, gender, address,
                         contact_number, password, registration_date,
                                 patient_flag, health_supporter_flag
Constraints: 1. The primary key is pid.
             2. The people_gender constraint ensures that the gender is valid.
             3. The people_pflag constraint ensures that the patient_flag is valid.
             4. The people_hflag constraint ensures that the health_supporter_flag is valid.
             5. The people_flag constraint ensures that the person is registering at least as a
             patient or health_supporter.
*/
CREATE TABLE people
    (pid VARCHAR(10) CONSTRAINT pk_people PRIMARY KEY,
    first_name VARCHAR(20),
    last_name VARCHAR(20),
    date_of_birth DATE,
    gender VARCHAR(1),
    address VARCHAR(50),
    contact_number NUMBER(10),
    password VARCHAR(20),
    registration_date DATE,
    patient_flag NUMBER(1),
    health_supporter_flag NUMBER(1),
    CONSTRAINT people_gender CHECK (gender IN ('m','M','F','f')),
    CONSTRAINT people_pflag CHECK (patient_flag IN (0,1)),
    CONSTRAINT people_hflag CHECK (health_supporter_flag IN (0,1)),
    CONSTRAINT people_flag CHECK (patient_flag = 1 OR health_supporter_flag = 1));


/*
Name: support
Description: Table contains relationship between patient and health supporter
Functional Dependencies: patient_pid and  health_supporter_pid functionally determine all other
columns.
                         patient_pid health_supporter_pid -> start_date end_date
                         health_supporter_type
Constraints: 1. (patient_pid, health_supporter_pid) is the PRIMARY KEY
             2. patient_pid and health_supporter_pid have a foreign key reference to people.
```

```sql
             3. The support_not_same constraint ensures that a patient is not designated as his
             own health supporter.
             4. The support_type constraint ensures that health_supporter_type is primary or
             secondary.
*/
CREATE TABLE support
    (patient_pid VARCHAR(10),
    health_supporter_pid VARCHAR(10),
    start_date DATE,
    end_date DATE,
    health_supporter_type VARCHAR(10),
    CONSTRAINT pk_support PRIMARY KEY (patient_pid, health_supporter_pid),
    CONSTRAINT fk_patient_pid FOREIGN KEY (patient_pid) REFERENCES people ON DELETE CASCADE ON
    UPDATE CASCADE,
    CONSTRAINT fk_health_supporter_pid FOREIGN KEY (health_supporter_pid) REFERENCES people ON
    DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT support_not_same CHECK (patient_pid <> health_supporter_pid),
    CONSTRAINT support_type CHECK (health_supporter_type IN ('primary', 'secondary')));


/*
Name: disease
Description: Table consists of the name of disease and a unique id associated with it.
Functional Dependencies: id functionally determines disease.
                         id -> disease
Constraints: 1. The primary key is id.
*/
CREATE TABLE disease
    (id NUMBER(10) CONSTRAINT pk_disease PRIMARY KEY,
    name varchar(20));


/*
Name: diagnosis
Description: Table contains diseases diagnosed for patients
Functional Dependencies: patient_pid and  disease_id functionally determine all other columns.
                         patient_pid disease_id -> diagnosis_date
Constraints: 1. (patient_pid, disease_id) is the PRIMARY KEY
             2. patient_pid has a foreign key reference to people.
             3. disease_id has a foreign key reference to disease.
*/
CREATE TABLE diagnosis
    (patient_pid VARCHAR(10),
    disease_id NUMBER(10),
    diagnosis_date DATE,
    CONSTRAINT pk_diagnosis PRIMARY KEY (patient_pid, disease_id),
    CONSTRAINT fk_diagnosis_patient_pid FOREIGN KEY (patient_pid) REFERENCES people ON DELETE
    CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_diagnosis_disease_id FOREIGN KEY (disease_id) REFERENCES disease ON DELETE
    CASCADE ON UPDATE CASCADE);


/*
Name: health_observation
Description: Table provides details about health observations such as unique id, description,
lower and upper limit.
Functional Dependencies: id functionally determines all other columns.
                         id -> name, description, data_type, lower_limit, upper_limit
Constraints: 1. The primary key is id.
             2. The lower_limit_pain_check constraint ensures that health observation of type
             pain is above 0.
             3. The upper_limit_pain_check constraint ensures that health observation of type
             pain is below 11.
*/
CREATE TABLE health_observation
    (id NUMBER(10) CONSTRAINT pk_health_observation PRIMARY KEY,
    name VARCHAR(20),
    description VARCHAR(50),
    data_type VARCHAR(10),
    lower_limit NUMBER(10),
    upper_limit NUMBER(10),
    CONSTRAINT lower_limit_pain_check CHECK (lower_limit > CASE WHEN data_type = 'pain' THEN 0
    END),
    CONSTRAINT upper_limit_pain_check CHECK (upper_limit < CASE WHEN data_type = 'pain' THEN 11
    END));

/*
Name: mood_mapping
Description: Table contains mapping between numerical and string values of mood.
Functional Dependencies: mood_number functionally determines all other columns.
                         mood_number -> mood_string
Constraints: 1. The primary key is mood_mapping
*/
CREATE TABLE mood_mapping
(mood_number number(10) CONSTRAINT pk_mood_mapping PRIMARY KEY,
mood_string varchar(10));

/*
Name: health_obs_frequency
Description: Table contains patient specific health observations and their frequency
Functional Dependencies: patient_pid and  health_obs_id functionally determine all other columns.
                         patient_pid health_obs_id -> frequency
Constraints: 1. (patient_pid, health_obs_id) is the PRIMARY KEY
             2. patient_pid has a foreign key reference to people.
             3. health_obs_id has a foreign key reference to health_observation.
*/
CREATE TABLE health_obs_frequency
    (patient_pid VARCHAR(10),
    health_obs_id NUMBER(10),
    frequency NUMBER(5),
    CONSTRAINT pk_health_obs_frequency PRIMARY KEY (patient_pid, health_obs_id),
    CONSTRAINT fk_health_obs_pid FOREIGN KEY (patient_pid) REFERENCES people ON DELETE CASCADE
    ON UPDATE CASCADE,
    CONSTRAINT fk_health_obs_id FOREIGN KEY (health_obs_id) REFERENCES health_observation ON
    DELETE CASCADE ON UPDATE CASCADE);


/*
Name: recorded_health_obs
Description: Table contains recorded values for every health observation of a patient. This
includes observed time and recorded time.
Functional Dependencies: rec_id functionally determines all other columns.
                         rec_id -> patient_id, health_obs_id, recorded_value, observed_time,
                         recorded_time
Constraints: 1. The primary key is rec_id.
             2. The fk_recorded_health_obs_pid constraint defines patient_pid as a foreign key
             reference to people.
             3. The fk_recorded_health_obs_id constraint defines health_obs_id as a foreign key
             reference to health_observation.
             4. The recorded_value_limit_1 constraint ensures that recorded_value for
             health_obs_id of value 6 (pain) is above 0.
             5. The recorded_value_limit_2 constraint ensures that recorded_value for
             health_obs_id of value 6 (pain) is below 11.
*/
CREATE TABLE recorded_health_obs
    (rec_id NUMBER(10),
    patient_pid VARCHAR(10),
    health_obs_id NUMBER(10),
    recorded_value NUMBER(10),
    observed_time DATE,
    recorded_time DATE,
    CONSTRAINT pk_recorded_health_obs PRIMARY KEY (rec_id),
```

```sql
    CONSTRAINT fk_recorded_health_obs_pid FOREIGN KEY (patient_pid) REFERENCES people ON DELETE
    CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_recorded_health_obs_id FOREIGN KEY (health_obs_id) REFERENCES
    health_observation ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT recorded_value_limit_1 CHECK (recorded_value > CASE WHEN health_obs_id = 6 THEN 0
     END),
    CONSTRAINT recorded_value_limit_2 CHECK (recorded_value < CASE WHEN health_obs_id = 6 THEN
    11 END));


/*
Name: disease_recommendations
Description: Table contains disease specific health observations and their frequency
Functional Dependencies: disease_id and  health_obs_id functionally determine all other columns.
                         disease_id health_obs_id -> frequency
Constraints: 1. (disease_id, health_obs_id) is the PRIMARY KEY
             2. disease_id has a foreign key reference to disease.
             3. health_obs_id has a foreign key reference to health_observation.
*/
CREATE TABLE disease_recommendations
    (disease_id NUMBER(10),
    health_obs_id NUMBER(10),
    frequency NUMBER(5),
    CONSTRAINT pk_disease_recommendations PRIMARY KEY (disease_id, health_obs_id),
    CONSTRAINT fk_disease_reco_id FOREIGN KEY (disease_id) REFERENCES disease ON DELETE CASCADE
    ON UPDATE CASCADE,
    CONSTRAINT fk_health_reco_obs_id FOREIGN KEY (health_obs_id) REFERENCES health_observation
    ON DELETE CASCADE ON UPDATE CASCADE);


/*
Name: patient_health_obs_limits
Description: Table contains patient specific limits for a health observation. This includes
lower and upper limits.
Functional Dependencies: patient_pid functionally determines all other columns.
                         patient_pid -> health_obs_id, lower_limit, upper_limit
Constraints: 1. The primary key is patient_pid.
             2. The fk_patient constraint defines patient_pid as a foreign key reference to
             people.
             3. The fk_health_obs constraint defines health_obs_id as a foreign key reference
             to health_observation.
             4. The obs_limits_lower_limit constraint ensures that lower_limit for
             health_obs_id of value 6 (pain) is above 0.
             5. The obs_limits_upper_limit constraint ensures that upper_limit for
             health_obs_id of value 6 (pain) is below 11.
*/
CREATE TABLE patient_health_obs_limits
    (patient_pid VARCHAR(10),
    health_obs_id NUMBER(10),
    lower_limit NUMBER(10),
    upper_limit NUMBER(10),
    CONSTRAINT pk_patient_health_obs_limits PRIMARY KEY (patient_pid, health_obs_id),
    CONSTRAINT fk_patient FOREIGN KEY (patient_pid) REFERENCES people ON DELETE CASCADE ON
    UPDATE CASCADE,
    CONSTRAINT fk_health_obs FOREIGN KEY (health_obs_id) REFERENCES health_observation ON DELETE
     CASCADE ON UPDATE CASCADE,
    CONSTRAINT obs_limits_lower_limit CHECK (lower_limit > CASE WHEN health_obs_id = 6 THEN 0
    END),
    CONSTRAINT obs_limits_upper_limit CHECK (upper_limit < CASE WHEN health_obs_id = 6 THEN 11
    END));

/*
Name: alert
Description: Table contains details about the types of alerts
Functional Dependencies: id  functionally determines all other columns.
                         id -> name
```

```sql
Constraints: 1. id is the PRIMARY KEY.
*/
CREATE TABLE alert
    (id NUMBER(10) CONSTRAINT pk_alerts PRIMARY KEY,
    name VARCHAR(30));

/*
Name: patient_alert_threshold
Description: Table contains threshold for each type of alert for every patient's health_obs_id.
Functional Dependencies: patient_pid functionally determines all other columns.
                         patient_pid -> health_obs_id, alert_id, threshold
Constraints: 1. The primary key is patient_pid.
             2. The fk_threshold_patient constraint defines patient_pid as a foreign key
             reference to people.
             3. The fk_threshold_health_obs constraint defines health_obs_id as a foreign key
             reference to health_observation.
             4. The fk_threshold_alert constraint defines alert_id as a foreign key reference
             to alert.
*/
CREATE TABLE patient_alert_threshold
    (patient_pid VARCHAR(10),
    health_obs_id NUMBER(10),
    alert_id NUMBER(10),
    threshold NUMBER(10),
    CONSTRAINT pk_patient_alert_threshold PRIMARY KEY (patient_pid, health_obs_id, alert_id),
    CONSTRAINT fk_threshold_patient FOREIGN KEY (patient_pid) REFERENCES people ON DELETE
    CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_threshold_health_obs FOREIGN KEY (health_obs_id) REFERENCES health_observation
     ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_threshold_alert FOREIGN KEY (alert_id) REFERENCES alert ON DELETE CASCADE ON
    UPDATE CASCADE);

/*
Name: recorded_alerts
Description: Table records all alerts
Functional Dependencies: rec_alert_id  functionally determines all other columns.
                         rec_alert_id -> patient_pid health_obs_id alert_id recorded_date
Constraints: 1. rec_alert_id is the PRIMARY KEY.
             2. patient_pid has a foreign key reference to people
             3. health_obs_id has a foreign key reference to health_observation
             4. alert_id has a foreign key reference to alert
*/
CREATE TABLE recorded_alerts
    (rec_alert_id NUMBER(10),
    patient_pid VARCHAR(10),
    health_obs_id NUMBER(10),
    alert_id NUMBER(10),
    recorded_date date,
    CONSTRAINT pk_recorded_alerts PRIMARY KEY (rec_alert_id),
    CONSTRAINT fk_recorded_alerts_patient FOREIGN KEY (patient_pid) REFERENCES people ON DELETE
    CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_recorded_alerts_health_obs FOREIGN KEY (health_obs_id) REFERENCES
    health_observation ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_recorded_alerts_alert FOREIGN KEY (alert_id) REFERENCES alert ON DELETE
    CASCADE ON UPDATE CASCADE);

/*
Name: seq
Description: Sequence numbers for generating primary key.
*/

CREATE SEQUENCE seq
    START WITH 1000
    INCREMENT BY 1
    NOCACHE
    NOCYCLE;
```

```
/*
==============================================================================================
CONSTRAINTS
==============================================================================================
*/


/*
Name: check_hsflag_in_people
Description: This trigger checks if a health supporter entered into the support table is
registered as a health supporter. If not registered, the health supporter is invalid.
*/

CREATE OR REPLACE TRIGGER check_hsflag_in_people
AFTER INSERT OR UPDATE ON support
FOR EACH ROW
DECLARE
row_nums NUMBER;
BEGIN
SELECT COUNT(*) INTO row_nums FROM people WHERE pid = :NEW.health_supporter_pid
                AND health_supporter_flag = 1;
IF row_nums = 0 THEN
 raise_application_error(-20001,'Invalid health supporter');
END IF;
END;
/

/*
Name: check_patientflag_in_people
Description: This trigger checks if a patient entered into the support table is
registered as a patient. If not registered, the patient is invalid.
*/

CREATE OR REPLACE TRIGGER check_patientflag_in_people
BEFORE INSERT OR UPDATE ON support
FOR EACH ROW
DECLARE
row_nums NUMBER;
BEGIN
SELECT COUNT(*) INTO row_nums FROM people WHERE pid = :NEW.patient_pid
            AND patient_flag = 1;
IF row_nums = 0 THEN
 raise_application_error(-20001,'Invalid patient');
END IF;
END;
/

/*
Name: hs_count_per_patient
Description: This trigger checks if a patient has two health supporters and
prevents the patient from having a third health supporter.
*/

CREATE OR REPLACE TRIGGER hs_count_per_patient
BEFORE INSERT OR UPDATE ON support
FOR EACH ROW
DECLARE
row_nums NUMBER;
BEGIN
SELECT COUNT(health_supporter_pid) INTO row_nums FROM support
        WHERE patient_pid = :NEW.patient_pid AND :NEW.health_supporter_pid IS NOT NULL;
IF row_nums = 2 THEN
 raise_application_error(-20001,'Cannot enter more than two health supporters');
```

```sql
END IF;
END;
/

/*
Name: hs_count_per_type_per_patient
Description: This trigger checks if a patient has one health supporter of each type
and prevents the patient from having more than one of each type. (Type refers to
primary/secondary)
*/

CREATE OR REPLACE TRIGGER hs_count_per_type_per_patient
BEFORE INSERT OR UPDATE ON support
FOR EACH ROW
DECLARE
primary_row_nums NUMBER;
secondary_row_nums NUMBER;
BEGIN
SELECT COUNT(health_supporter_pid) INTO primary_row_nums FROM support
        WHERE patient_pid = :NEW.patient_pid AND health_supporter_type = 'primary';
IF primary_row_nums = 1 AND :NEW.health_supporter_type = 'primary' THEN
 raise_application_error(-20001,'cannot have more than one primary health supporter');
END IF;
SELECT COUNT(health_supporter_pid) INTO secondary_row_nums FROM support
        WHERE patient_pid = :NEW.patient_pid AND health_supporter_type = 'secondary';
IF secondary_row_nums = 1 AND :NEW.health_supporter_type = 'secondary' THEN
 raise_application_error(-20001,'Cannot have more than one secondary health supporter');
END IF;
END;
/

/*
Name: primary_hs_existence_check
Description: This trigger checks if a patient has a primary health supporter
and prevents the patient entering a secondary health supporter if a primary health supporter
does not exist.
*/

CREATE OR REPLACE TRIGGER primary_hs_existence_check
BEFORE INSERT OR UPDATE ON support
FOR EACH ROW
DECLARE
row_nums NUMBER;
BEGIN
SELECT COUNT(health_supporter_pid) INTO row_nums FROM support
        WHERE patient_pid = :NEW.patient_pid AND health_supporter_type = 'primary';
IF row_nums = 0 AND :NEW.health_supporter_type = 'secondary' THEN
 raise_application_error(-20001,'Primary health supporter does not exist. Please choose health
 supporter type as "primary"');
END IF;
END;
/


/*
Name: update_sec_to_primary
Description: This function deletes the primary health supporter and converts the
secondary health supporter to a primary health supporter.
*/

CREATE OR REPLACE FUNCTION update_sec_to_primary (pat_id VARCHAR2)
  RETURN NUMBER IS
  rows_num NUMBER;
  BEGIN
  SELECT COUNT(*) INTO rows_num FROM support
                WHERE PATIENT_PID = pat_id AND health_supporter_type = 'secondary';
  DELETE FROM support WHERE PATIENT_PID = pat_id AND health_supporter_type = 'primary';
  IF rows_num = 0
    THEN RETURN 0;
  ELSE
    EXECUTE IMMEDIATE 'ALTER TRIGGER check_patientflag_in_people DISABLE';
    EXECUTE IMMEDIATE 'ALTER TRIGGER hs_count_per_patient DISABLE';
    EXECUTE IMMEDIATE 'ALTER TRIGGER check_hsflag_in_people DISABLE';
    EXECUTE IMMEDIATE 'ALTER TRIGGER hs_count_per_type_per_patient DISABLE';
    EXECUTE IMMEDIATE 'ALTER TRIGGER primary_hs_existence_check DISABLE';

    UPDATE support
      SET health_supporter_type = 'primary'
      WHERE patient_pid = pat_id;
      EXECUTE IMMEDIATE 'ALTER TRIGGER check_hsflag_in_people ENABLE';
      EXECUTE IMMEDIATE 'ALTER TRIGGER check_patientflag_in_people ENABLE';
      EXECUTE IMMEDIATE 'ALTER TRIGGER hs_count_per_patient ENABLE';
      EXECUTE IMMEDIATE 'ALTER TRIGGER hs_count_per_type_per_patient ENABLE';
      EXECUTE IMMEDIATE 'ALTER TRIGGER primary_hs_existence_check ENABLE';

    RETURN 1;

  END IF;
  END;
  /


  /*
Name: happy_mapping
Description: This function is called when a values are entered for a patient.
If the entry is a mood, a mapping is done to convert it to a numeric equivalent.
*/

CREATE OR REPLACE FUNCTION happy_mapping (pat VARCHAR, health_obs NUMBER,
                    recorded_value VARCHAR, obs_time TIMESTAMP, rec_time TIMESTAMP)
RETURN NUMBER IS
flag NUMBER(10);
rec_value NUMBER(10);
BEGIN
  IF (health_obs = 7) THEN
    IF recorded_value = 'happy' THEN rec_value := 1;
    ELSIF recorded_value = 'neutral' THEN rec_value := 2;
    ELSIF recorded_value = 'sad' THEN rec_value := 3;
    END IF;
    INSERT INTO recorded_health_obs
            VALUES (seq.NEXTVAL, pat, health_obs, rec_value, obs_time, rec_time);
    SELECT COUNT(*) INTO flag FROM recorded_health_obs WHERE patient_pid = pat
                  AND health_obs_id = health_obs AND recorded_value = rec_value
                  AND observed_time = obs_time
                  AND recorded_time = rec_time;
  ELSE
    INSERT INTO recorded_health_obs
            VALUES (seq.NEXTVAL, pat, health_obs, recorded_value,obs_time,rec_time);
    SELECT COUNT(*) INTO flag FROM recorded_health_obs WHERE patient_pid = pat
                  AND health_obs_id = health_obs AND recorded_value = recorded_value
                  AND observed_time = obs_time
                  AND recorded_time = rec_time;
  END IF;
  RETURN flag;
END;
/


/*
Name: check_sick
Description: When passed a pid for a patient, this function determines if the
patient is sick or well by checking if there is any row in the diagnosis table for that patient.
```

```
*/
CREATE OR REPLACE FUNCTION check_sick (p_pid IN VARCHAR)
RETURN NUMBER IS
sick_count NUMBER(1);
BEGIN
    SELECT COUNT(*) into sick_count
    FROM diagnosis
    WHERE diagnosis.patient_pid = p_pid;
    RETURN sick_count;
END;
/


/*
Name: sick_needs_health_supporter
Description: When passed pid of a patient, this function determines if the
patient is sick and doesn't have a health supporter. This function is
called everytime a patient logs in. If it is found that the patient is sick
and doesn't have a health supporter, patient is promted to add a health supporter.
*/
CREATE OR REPLACE FUNCTION sick_needs_health_supporter (p_pid IN VARCHAR)
RETURN NUMBER IS
support_flag NUMBER(1);
sick_count NUMBER (1);
support_count NUMBER (1);
BEGIN
  SELECT COUNT(*) into sick_count
    FROM diagnosis
    WHERE diagnosis.patient_pid = p_pid;
  SELECT COUNT(*) into support_count
    FROM support
    WHERE support.patient_pid = p_pid;
    IF(sick_count>0 AND support_count<1) THEN
    support_flag := 1;--Need to add health supporter
  ELSE
    support_flag := 0;--No need for health supporter
  END IF;
    RETURN support_flag;
END;
/


/*
Name: validate_patient
Description: When passed the pid and password of a patient at the time of log in,
this function determines if the credentials are correct
*/
CREATE OR REPLACE FUNCTION validate_patient (upid IN VARCHAR, upassword  IN VARCHAR)
RETURN INT IS
pcount INT;
BEGIN
    SELECT COUNT(*) INTO pcount
    FROM people
    WHERE pid = upid
    AND password = upassword
    AND patient_flag = 1;
    RETURN pcount;
END;
/


/*
Name: validate_health_support
Description: When passed the pid and password of a health supporter at the
time of log in, this function determines if the credentials are correct
*/
```

```
CREATE OR REPLACE FUNCTION validate_health_support (upid IN VARCHAR, upassword  IN VARCHAR)
RETURN INT IS
hcount INT;
BEGIN
    SELECT COUNT(*) INTO hcount
    FROM people
    WHERE pid = upid
    AND password = upassword
    AND health_supporter_flag = 1;
    RETURN hcount;
END;
/


/*
Name: outside_limit_alert
Description: When passed the pid and corresponding health_observation.id of a patient,
it first determines the appropriate upper and lower limits for the patient's health observation.
It then checks if any of the recorded values are beyond the specified limits,
and inserts alerts into recorded_alerts table.
*/
CREATE OR REPLACE PROCEDURE outside_limit_alert (p_pid IN VARCHAR, p_hid IN NUMBER)
IS
my_lower_limit NUMBER(10);
my_upper_limit NUMBER(10);
temp_count NUMBER(10);
CURSOR c1  IS
SELECT recorded_value, recorded_time
FROM recorded_health_obs
WHERE patient_pid = p_pid
AND health_obs_id = p_hid;
my_recorded_value c1%ROWTYPE;
BEGIN
 SELECT COUNT(*) INTO temp_count
 FROM patient_health_obs_limits
 WHERE patient_pid = p_pid
 AND health_obs_id = p_hid;
 IF (temp_count = 1) THEN
  SELECT lower_limit INTO my_lower_limit
  FROM patient_health_obs_limits
  WHERE patient_pid = p_pid
  AND health_obs_id = p_hid;
  SELECT upper_limit INTO my_upper_limit
  FROM patient_health_obs_limits
  WHERE patient_pid = p_pid
  AND health_obs_id = p_hid;
 ELSE
  SELECT lower_limit INTO my_lower_limit
  FROM health_observation
  WHERE id = p_hid;
  SELECT upper_limit INTO my_upper_limit
  FROM health_observation
  WHERE id = p_hid;
 END IF;
 OPEN c1;
  LOOP
   FETCH c1 INTO my_recorded_value;
   EXIT WHEN c1%NOTFOUND;
   IF(my_recorded_value.recorded_value < my_lower_limit
            OR my_recorded_value.recorded_value > my_upper_limit) THEN
    INSERT INTO recorded_alerts (rec_alert_id, patient_pid, health_obs_id, alert_id,
    recorded_date)
                VALUES (seq.nextVal, p_pid, p_hid, 1, my_recorded_value.recorded_time);
   END IF;
  END LOOP;
 CLOSE c1;
```

```
END;
/

/*
Name: low_activity_alert
Description: When passed the pid, corresponding health_observation.id  and start date,
it determines the corresponding frequency and checks if health observation was recorded the
required
number of times. For any missed health_observation it inserts corresponding alerts
into recorded_alerts table. Here, start date is the diagnosis date if health observation is being
recorded for a diagnosis. If health observation is patient specific,
then start date is the registration date of the patient.
*/
CREATE OR REPLACE PROCEDURE low_activity_alert(p_pid IN VARCHAR, p_hid IN NUMBER, p_date IN OUT
DATE)
IS
my_frequency NUMBER(10);
my_count NUMBER(10);
BEGIN
 SELECT min(frequency) INTO my_frequency
 FROM diagnosis dia, disease_recommendations dr
 WHERE dia.patient_pid = p_pid
 AND dia.disease_id = dr.disease_id
 AND dr.health_obs_id = p_hid;
 WHILE p_date + my_frequency < SYSDATE ---Full interval completed
 LOOP
 SELECT COUNT(*) INTO my_count
 FROM recorded_health_obs
 WHERE patient_pid = p_pid
 AND health_obs_id = p_hid
 AND recorded_time >= p_date
 AND recorded_time < p_date + my_frequency;
  p_date := p_date+my_frequency;
 IF(my_count = 0) THEN
    INSERT INTO recorded_alerts (rec_alert_id, patient_pid, health_obs_id, alert_id,
    recorded_date)
            VALUES (seq.nextVal, p_pid, p_hid, 2, p_date+my_frequency-1);
--Insert end date of interval, health support that was assigned any time within the interval
can still see the alert
 END IF;
 END LOOP;
END;
/
```