Database Theory and Applications for Biomedical Research and Practice (BMIN 502/EPID 635)
Spring 2019
Assignment 9: Working with the Entity-Attribute-Value format, Stored routines, and Triggers

Part A

1. Import the patient list file in EAV format into a schema (the .csv file is in the Files folder on Canvas)

2. Create a data dictionary for the schema you just created

3. Create a query that transposes the data from an EAV format to the relational table format

4. Create a data dictionary for the schema you just updated

5. Parse the results to strip the text and leave only the numeric value

6. Queries (you have a lot of freedom here, but you should write three)
    a. Obtain the mean hemoglobin for all patients
    b. Obtain the mean platelets separately for males and females

7. Submit your .sql code and the result of your queries in Step 6 to Assignment 9 as: **yourlastname_BMIN502_19_9a.pdf**

Part B
Stored routines
Stored routines are a way to modularize and centrally store operations that can be invoked any time and in any database. A stored routine facilitates code reuse, in that you can pass parameters to it from a SQL script, where the parameters take on values specific to a given database. You can create a stored routine as either a procedure of a function, using the commands CREATE PROCEDURE or CREATE FUNCTION, respectively. A stored routine is associated with the database in which it was created, but you can reference the routine in another database as `database_name.routine_name`. Stored routines can take parameters, which are variables that take on a value to be tested or otherwise used in the routine. In this way, these are like the functions you have already seen, like SUBSTR(), where what goes into the parentheses are parameters (name of string, starting position, characters to copy).

Here is an example of a stored procedure in SQL. It restricts the selection of records where gender is equal to whatever value is passed to it by the `CALL` statement below.

```
DELIMITER //                                         # Enables the ; to be ignored in procedure
CREATE PROCEDURE gender_restrict                     # Creates procedure named restrict_by_gender
     (IN gender_param CHAR(10))                      # Tests for contents of the passed parameter
BEGIN                                                # Begin procedure code
     SELECT patientID, age FROM Demographics         # Here's the procedure
     WHERE gender = gender_param;                    # Note that the parameter is evaluated here
END //                                               # End procedure code
DELIMITER ;                                          # Change delimiter back to ;
```

And it would be called as:

```
     CALL gender_restrict('Female');                 # Note that 'Female' is the parameter passed
                                                     # to the procedure. It could # be any value,
                                                     like 'Male', # '1'- whatever would be a
                                                     # legal value for gender in
                                                     # the database.
```

Write a stored procedure of your choice for your ABIC database, and a query that calls it. Submit your .sql code and the result of your query to Assignment 9 as:
yourlastname_BMIN502_19_9b.pdf

Part B, continued
Triggers
Triggers are objects that are activated when some specified condition is met, usually through an action or process performed on a table. Typical processes or actions include `INSERT`, `DELETE`, and `UPDATE`.

Here is an example of a trigger in SQL. Here we have a Demographics table with just three fields: patientID, age, and gender. We want to make sure no negative ages are added to a table, Demographics. Any attempt to add a negative age will cause an age of 0 to be inserted instead

```
DELIMITER //                                 # Enables the ; to be ignored in trigger
CREATE TRIGGER agecheck BEFORE INSERT        # Creates procedure named age_check
     ON Demographics                         # Sets up the test before inserting
     FOR EACH ROW                            # Loops through each row in Demographics
          IF NEW.age < 0 THEN                # Evaluates parameter value
               SET NEW.age = 0;              # If the parameter value <0, set it to 0
          END IF;//                          # Completes evaluation
DELIMITER ;                                  # Change delimiter back to ;
```

And here is how it would be called:
`INSERT INTO` Demographics `VALUES` (‘1’, -5, ‘Male’), (‘2’, 21,’Female’), (‘3’,30,’Female’);

The result will be:

patientID	Age	gender
1	0	Male
2	21	Female
3	30	Female

Write a trigger of your choice for your ABIC database, and a query that causes the trigger to be fired. Submit your .sql code and the result of your query to Assignment 9 as:
yourlastname_BMIN502_19_9c.pdf