Nick Carrozza

CMPT 308

Database Final Project Report

08 December 2016

A comprehensive, fully functional relational database for:

Developed, Designed, and Written by:
Nick Carrozza

Executive Summary:

Included in this report is a fully functional, comprehensive database written in SQL and implemented in PostgreSQL v9.5 for the Disney Pixar movie "Monsters Inc." This database follows the standard method of normalization and is in third-normal form.

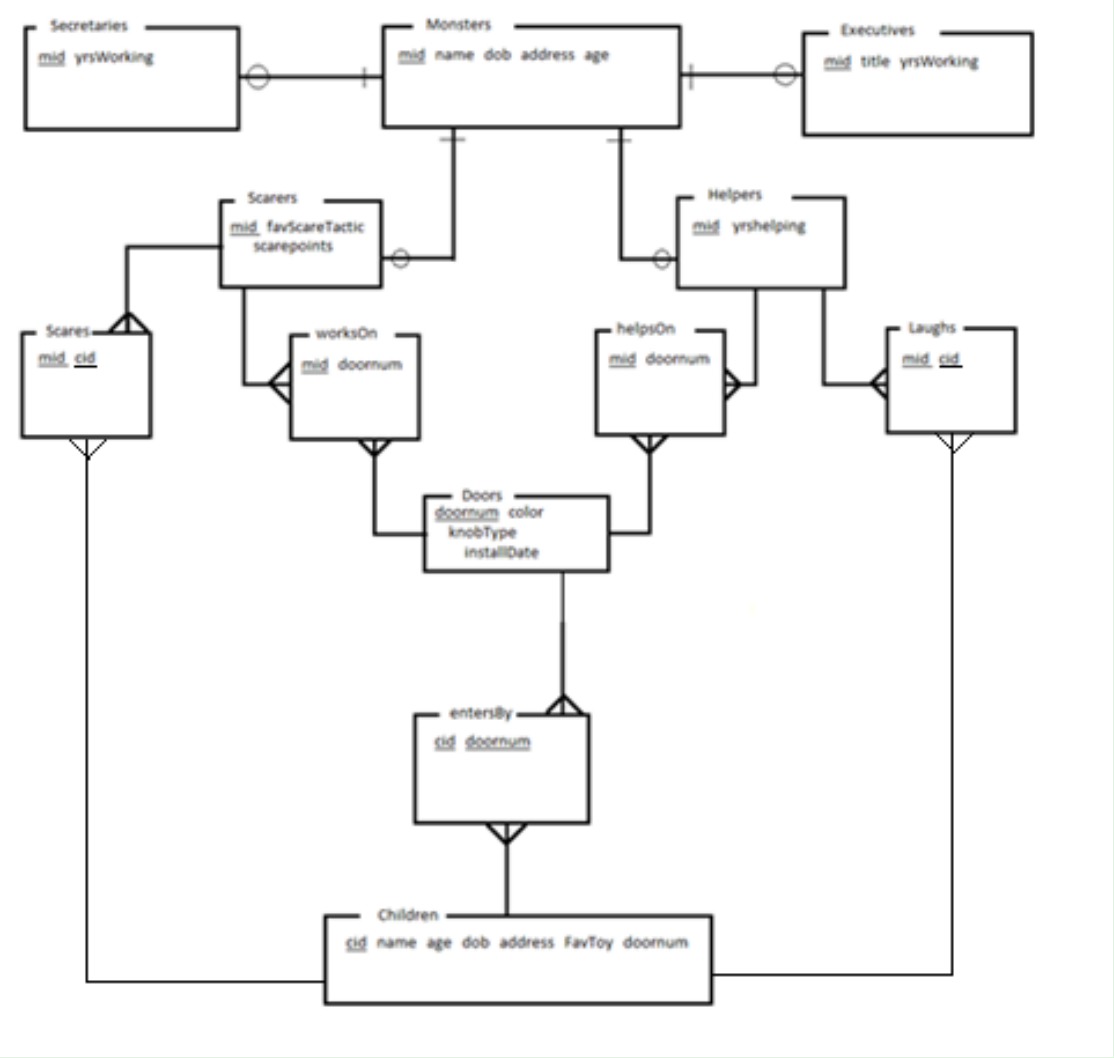
More specifically, the database serves to represent information pertaining to the fictional corporation "Monsters Incorporated" as it exists in the film. Several tables are instantiated, whose relationships reflect the information given in the movie to the greatest possible accuracy. In addition, the included database reflects and documents the roles, behaviors and relationships of characters and the activities these characters engage in. Despite this, some fictitious data is written and included for both testing purposes and the purpose of demonstration.

Objectives for this database include the following:

- To provide information regarding monsters, children, doors, etc. that exist in the Monsters Inc. world
- To outline the relationships between these entities and how to retrieve useful information about these relationships
- To create various views and stored procedures in an effort to increase usability of the database
- Define various roles that exist in the database and what privileges exist for them
- Create stored procedures in an effort to assist users of the database in retrieving useful information

ER Diagram:

The following shows an entity relationship diagram that demonstrates the relationships between the entities in the database. All primary keys are underlined in their respective tables.



Tables:

Monsters Table:

A "monsters" table is written as an entity supertype to the various roles that monsters exhibit in the movie. This table documents several important characteristics that apply to all monsters regardless of role in the company.

Monsters Table Dependencies:

mid → name, dob, address, age

SQL Code to create Monsters table:

```
CREATE TABLE monsters (
        mid             int             not null  unique,
        name            char(40)        not null,
        age             int             not null,
        dob             date            not null,
        address         char(50)        not null,

                primary key(mid)
);
```

Scarers Table:

A Scarers table is implemented one of the entity subtypes to the monsters table. This table includes monsters that are considered scarers at Monsters Inc., (e.g. Sully and Randall are scarers on the "scare floor" at Monsters Inc.).

Scares Table Dependencies:

mid → FavScareTactic, scarepoints

SQL Code to create Scarers table:

```
CREATE TABLE scarers (
        mid                 int             not null unique,
        FavScareTactic      char(50)        not null,
        scarepoints         int ,
                primary key(mid)
);
```

Helpers Table:

A helpers table is included as an entity subtype of the monsters table. These include monsters that help the scarers on the "scare floor" (i.e. Mike Wazowski is a helper to Sully the scarer).

Helpers Table Dependencies:

mid → yrsHelping

SQL Code to create Helpers Table:

```
CREATE TABLE helpers (
        mid             int     not null unique,
        yrsHelping      int     not null,
                primary key(mid)
);
```

Secretaries Table:

A table called Secretaries is created as another entity subtype to the Monsters table.

Secretaries Table Dependencies:

mid → yrsWorking

SQL Code to create the Secretaries table:

```
CREATE TABLE secretaries (
        mid             int     not null unique,
        yrsWorking      int     not null,
                primary key(mid)
);
```

Executives Table:

As the final entity subtype to the Monsters supertype, an executives table is created.

Executives Table Dependencies:

mid → title, yrsWorking

SQL Code to create the Executives table:

```
CREATE TABLE executives  (
        mid             int             not null unique,
```

```
        title           char(40)        not null,
        yrsWorking      int             not null,
                primary key(mid)
    );
```

worksOn Table:

As the first associative entity instantiated in the database, this table is necessary to link the "Scarers" and "Doors" tables together, allowing the relationship between the two of them to occur. The design of this table allows any given scarer to work on any number of doors, while also allowing multiple scares to work on the same door thereby satisfying the many-to-many relationship between these two entities.

worksOn Table Dependencies:

mid, doornum →

SQL Code to write the worksOn Table:

```
    CREATE TABLE worksOn (
        mid             int     not null,
        doornum         int     not null,
                primary key(mid, doornum)
    );
```

helpsOn Table:

The helpsOn table is created to provide the relationship between helpers and doors, and documents all the doors a given helper has worked on, or all the helpers that worked on a particular door. With fields identical to the

"worksOn" table, this allows many helpers to help on many doors, and satisfies this relationship in a relational context.

helpsOn Table Dependencies:

mid, doornum →

SQL Code to create helpsOn table:

```
CREATE TABLE helpsOn (
        mid             int     not null,
        doornum         int     not null,
                primary key(mid, doornum)
);
```

Scares Table:

A scares table is created to provide a relationship between scarers and children. This relationship literally entails a scarer "scaring" a child, where a scarer could scare many children. As outlined in the movie, typically there is only one monster assigned to each child. However, in an effort to keep the database relational and in 3NF throughout all-time, it has been deemed a many-to-many relationship , with the scares table serving as the associative entity between scarers and children. That way, in the event that a child is scared by more than one monster, this event can be supported by the database.

Scares Table Dependencies:

mid, cid →

SQL Code to create Scares Table:

```
CREATE TABLE scares (
        mid     int     not null,
```

```
        cid     int     not null,
                primary key(mid, cid)
    );
```

Laughs Table:

A laughs table is created in order to satisfy the relationship between the helpers and the children they make laugh. Different from the helpsOn table, the laughs table serves the role of helpers as "comedians" where, at the end of the movie, helpers no longer help scarers. Instead, they make the children laugh rather than help the scarers scare the children. With similar fields to the scares table, the laughs table provides a like to the children's table from the helpers table, which is necessary to fulfill the new role of the helpers.

Laughs Table Dependencies:

mid, cid →

SQL Code to create laughs table:

```
    CREATE TABLE laughs (
        mid     int     not null,
        cid     int     not null,
                primary key(mid, cid)
    );
```

Children Table:

An additional strong entity is created with the children table. Here, the characteristics of the children in the movie are outlined. It is important to recognize that the children table has a one to many relationship with both the laughs and scares tables, which provide the relationship of the children to the helpers and scarers. Also, doornum is included as a foreign key, which is referenced as the primary key of the doors table.

Children Table Dependencies:

cid → name, age, dob, address, FavToy, doornum

SQL Code to create children table:

```
CREATE TABLE children (
        cid             int             not null unique,
        name            char(40)        not null,
        age             int             not null,
        dob             date            not null,
        address         char(50)        not null,
        FavToy          char(30),
        doornum         int             not null,
                primary key(cid)
);
```

Doors Table:

Another strong entity, the doors table, is created to document all the doors that exist at Monsters Inc.

Doors Table Dependencies:

doornum → color, knobtype, installDate

SQL Code to create Doors table:

```
CREATE TABLE doors (
        doornum         int         not null unique,
        color           char(40)    not null,
        knobType        char(15)    not null,
        installDate     date        not null,
                primary key(doornum)
);
```

entersBy Table:

The entersBy table is an associative entity that links the children and doors tables. The design of this table allows more than one door to be assigned to a particular child. It also allows for multiple children to share a door, in the event that more than one child shares the same room.

entersBy Table Dependencies:

cid, doornum →

SQL Code to create entersBy table:

```
CREATE TABLE entersBy (
        cid     int     not null,
        doornum int     not null,
                primary key(cid, doornum)
```

);

Security: Grant and Revoke Privileges:

This database accounts for three different roles. Adminstrators, Point Counters, and Executives roles are created, where each role has certain privileges granted to them when they access information from the database.

Administrators Role:

The administrator is allowed access to all privileges, or commands, as well as full access of the database. This is necessary so they have the proper autonomy and control over the database to make necessary changes and to ensure that it works properly.

SQL Code for GRANT and REVOKE Privileges on Administrator Role:

```
GRANT SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER ON SullysHelpers TO Admins;

GRANT SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER ON RandallsPoints TO Admins;

GRANT SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER ON monsters TO Admins;

GRANT SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER ON helpers TO Admins;

GRANT SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER ON scarers TO Admins;

GRANT SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER ON secretaries TO Admins;

GRANT SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER ON executives TO Admins;
```

GRANT SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER ON laughs TO Admins;

GRANT SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER ON helpsOn TO Admins;

GRANT SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER ON worksOn TO Admins;

GRANT SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER ON scares TO Admins;

GRANT SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER ON doors TO Admins;

GRANT SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER ON entersBy TO Admins;

GRANT SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER ON children TO Admins;

Point Counters Role:

Point Counters are responsible for the accurate counting of the points scored by each monster on the Monsters Inc. “scare floor” as it exists in the movie. This role is allowed only to update the scarers table where the “scarepoints” column exists. All other privileges for this role have been revoked.

SQL Code for GRANT and REVOKE Privileges on Point Counters Role:

GRANT SELECT, UPDATE ON scarers TO PointCounters;

REVOKE INSERT, DELETE, TRUNCATE, REFERENCES, TRIGGER ON scarers FROM PointCounters;

REVOKE ALL PRIVILEGES ON monsters FROM PointCounters;

REVOKE ALL PRIVILEGES ON helpers FROM PointCounters;

REVOKE ALL PRIVILEGES ON secretaries FROM PointCounters;

```
REVOKE ALL PRIVILEGES ON executives FROM PointCounters;

REVOKE ALL PRIVILEGES ON laughs FROM PointCounters;

REVOKE ALL PRIVILEGES ON helpsOn FROM    PointCounters;

REVOKE ALL PRIVILEGES ON worksOn FROM     PointCounters;

REVOKE ALL PRIVILEGES ON scares FROM PointCounters;

REVOKE ALL PRIVILEGES ON doors FROM PointCounters;

REVOKE ALL PRIVILEGES ON entersBy FROM PointCounters;

REVOKE ALL PRIVILEGES ON children FROM PointCounters;
```

Executives Role:

As implemented in the database, the executives supervise the employees at Monsters Inc. and are therefore allowed to access certain aspects of the database. This includes the ability to select all aspects of the database. Despite this, executives cannot alter the data in any way, as that is reserved for administrators. This restriction is reserved for the Administrator role.

SQL Code for GRANT and REVOKE Privileges on Executives Role:

```
GRANT SELECT ON SullysHelpers TO Execs;

GRANT SELECT ON RandallsPoints TO Execs;

GRANT SELECT ON monsters TO Execs;

GRANT SELECT ON helpers TO Execs;

GRANT SELECT ON scarers TO Execs;

GRANT SELECT ON secretaries TO Execs;

GRANT SELECT ON executives TO Execs;

GRANT SELECT ON laughs TO Execs;
```

GRANT SELECT ON helpsOn TO Execs;

GRANT SELECT ON worksOn TO Execs;

GRANT SELECT ON scares TO Execs;

GRANT SELECT ON doors TO Execs;

GRANT SELECT ON entersBy TO Execs;

GRANT SELECT ON children TO Execs;

REVOKE UPDATE, INSERT, DELETE, TRUNCATE, REFERENCES, TRIGGER ON SullysHelpers TO Execs;

REVOKE UPDATE, INSERT, DELETE, TRUNCATE, REFERENCES, TRIGGER ON RandallsPoints TO Execs;

REVOKE UPDATE, INSERT, DELETE, TRUNCATE, REFERENCES, TRIGGER ON executives TO Execs;

REVOKE UPDATE, INSERT, DELETE, TRUNCATE, REFERENCES, TRIGGER ON monsters TO Execs;

REVOKE UPDATE, INSERT, DELETE, TRUNCATE, REFERENCES, TRIGGER ON helpers TO Execs;

REVOKE UPDATE, INSERT, DELETE, TRUNCATE, REFERENCES, TRIGGER ON scarers TO Execs;

REVOKE UPDATE, INSERT, DELETE, TRUNCATE, REFERENCES, TRIGGER ON secretaries TO Execs;

REVOKE UPDATE, INSERT, DELETE, TRUNCATE, REFERENCES, TRIGGER ON laughs TO Execs;

REVOKE UPDATE, INSERT, DELETE, TRUNCATE, REFERENCES, TRIGGER ON helpsOn TO Execs;

REVOKE UPDATE, INSERT, DELETE, TRUNCATE, REFERENCES, TRIGGER ON worksOn TO Execs;

REVOKE UPDATE, INSERT, DELETE, TRUNCATE, REFERENCES, TRIGGER ON scares TO Execs;

REVOKE UPDATE, INSERT, DELETE, TRUNCATE, REFERENCES, TRIGGER ON doors TO Execs;

REVOKE UPDATE, INSERT, DELETE, TRUNCATE, REFERENCES, TRIGGER ON entersBy TO Execs;

REVOKE UPDATE, INSERT, DELETE, TRUNCATE, REFERENCES, TRIGGER ON children TO Execs;

Views:

Various views are written for the database, all of which provide a certain degree of usefulness to different users that interact with the database.

SullysHelpers View:

A view is created that displays the names of all the helpers that have helped "Sully", one of the main characters in Monsters Inc., has worked with on the scare floor. This is done with a where clause on the name equal to "Sully". Of course, this is done with an important assumption that there are no other monsters that share Sully's name. It would be possible to update this view such that, in the event of a repeated name, the where clause restricts by "mid" as the primary key and not by the candidate key "name", as it may, in the future, no longer be a candidate key.

SQL Code to create SullysHelpers View:

```
CREATE VIEW SullysHelpers AS
        SELECT name
        FROM monsters
        WHERE mid IN (
                SELECT helpsOn.mid
                FROM helpsOn
                INNER JOIN children ON helpsOn.doornum
=children.doornum
```

```
INNER JOIN worksOn ON worksOn.doornum =
children.doornum

INNER JOIN scarers ON scarers.mid = worksOn.mid

INNER JOIN monsters ON monsters.mid = scarers.mid

        WHERE monsters.name = 'Sully'

);
```

RandallsPoints View:

A view is created that retrieves the points that one of the main characters in Monsters Inc., Randall, has accumulated on the scare floor. A coalesce function is included to display a score of '0' in the event that no points exist in that column (important, although unlikely).

SQL Code to create RandallsPoints View:

```
CREATE VIEW RandallsPoints AS

    SELECT coalesce(scarepoints, '0')

    FROM scarers

    WHERE mid IN (    SELECT mid

                      FROM monsters

                      WHERE name = 'Randall'

                  );
```

MostScares View:

A view is created to display the monster with the most scares in the database, thus, the monster that has scared the most children. This could be particularly useful to the executives in determining who's performing the best, for example. This query uses a count function on the scares table, particularly the "mid" column determine the number of times a given "mid" appears in the scares table. A few attempts to write a sub-query proved ineffective, where inner joins were necessary to write this view.

SQL Code to create MostScares View:

```
CREATE VIEW MostScares AS

  SELECT monsters.name, scares.mid, count(scares.mid)
      AS qty_children_scared
      FROM scares
      INNER JOIN scarers ON scarers.mid = scares.mid
      INNER JOIN monsters ON monsters.mid = scarers.mid
          GROUP BY monsters.name, scares.mid
          ORDER BY qty_children_scared desc
              LIMIT 1;
```

Stored Procedures:

Stored procedures are helpful in executing a query more than once, so that the user does not have to write the same query over and over. Instead, they can simply call the stored procedure, which is a function that takes parameters specific to what the user needs.

show_scarers_helpers Stored Procedure:

The show_scarers_helpers stored procedure is useful for retrieving a helper who works with a given scarer, passed in the function as a parameter. This stored procedure uses the scarers name and returns a corresponding helper or helpers that that given scarer has worked with.

SQL Code to create show_scarers_helpers Stored Procedure:

```
CREATE FUNCTION show_scarers_helpers (text, refcursor)
        RETURNS refcursor AS
        $$
        DECLARE
                thisScarers_name    text            := $1;
                thisResult          refcursor       := $2;
        BEGIN
                OPEN thisResult for
                        SELECT name
                        FROM monsters
                        WHERE mid IN (  SELECT helpsOn.mid
                                        FROM helpsOn
                                        INNER JOIN children ON
                                        helpsOn.doornum =
                                        children.doornum
                                        INNER JOIN worksOn ON
                                        worksOn.doornum =
                                        children.doornum
```

```
                                        INNER JOIN scarers ON
                                        scarers.mid = worksOn.mid

                                        INNER JOIN monsters ON
                                        monsters.mid = scarers.mid

                                                WHERE monsters.name =
                                                thisScarers_name

                                        );

                RETURN thisResult;

        END;

        $$

        LANGUAGE PLPGSQL;


-- To show all the helpers "Sully" (a scarer) has worked with

        SELECT show_scarers_helpers ('Sully', 'results');

        FETCH ALL FROM results;


-- To show all the helpers "Randall" (a scarer)  has worked with

        SELECT show_scarers_helpers ('Randall', 'results');

        FETCH ALL FROM results;

-- To show all the helpers "Bile" (a scarer) has worked with

        SELECT show_scarers_helpers ('Bile', 'results');

        FETCH ALL FROM results;
```

made_children_laugh Stored Procedure:

This stored procedure is written to show all the monsters that made a given child, whose name is passed into the function as a parameter, laugh.

SQL Code to create made_children_laugh Stored Procedure:

```
CREATE FUNCTION made_children_laugh (text, refcursor)
        RETURNS refcursor AS
        $$
        DECLARE
                thisChilds_name     text            :=  $1;
                thisResult          refcursor       :=  $2;
        BEGIN
                OPEN thisResult for

                SELECT monsters.name
                FROM monsters
                INNER JOIN helpers ON helpers.mid = monsters.mid
                INNER JOIN laughs ON laughs.mid = helpers.mid
                INNER JOIN children ON children.cid = laughs.cid
                        WHERE children.name = thisChilds_name;
                RETURN thisResult;
        END;
        $$
        LANGUAGE PLPGSQL;
```

```
--To show all the monsters that made child 'Lily' laugh

    SELECT made_children_laugh ('Lily' , 'results');

    FETCH all from results;

--To show all the monsters that made child 'Sam' laugh

    SELECT made_children_laugh ('Sam' , 'results');

    FETCH all from results;

--To show all the monsters that made child 'Billy' laugh

    SELECT made_children_laugh ('Billy' , 'results');

    FETCH all from results;
```

Known Problems and Future Enhancements:

One possible issue with the database that may arise in the future is the fact that two of the stored procedures take in “names” of children and monsters as parameters. Because of this, if two monsters or children have the same name, then output will be returned with both of the entities with the same name, not the targeted one.

Another possible future enhancement could include deleting the “scares” associative entity from the database, as it is not necessary to define the relationship between monsters that scare children and the children themselves. This table does demonstrate that, since it is possible that two children could share the same door, that this relationship ultimately linking the tables together is unfavorable and should be accounted for otherwise.