Utility Management Database

Database Systems (CSS 475)

Professor: Dr. Erika Parsons

Semester: Fall 2015

Team Name: The Street Sharks

Members: Chris DuBois, Brody Schulke, Robert Stachofsky, Phat Tran

Project Proposal

Iteration III

Introduction

Managing and maintaining city utilities is essential for ensuring that the city runs smoothly and every citizen has access to the everyday resources that make lives easier. As cities continue to grow and utility systems become more complex, managing these utilities become ever-more complicated. While it is still feasible for companies to manually manage the needed repairs for various utilities throughout cities, there is a much easier way. We propose a city utility management database to assist utility management and repair staff in effortlessly being able to identify and locate their responsibilities as well as scheduling and tracking various tasks throughout the city. In essence, this utility management database system will increase productivity throughout organizations, and ensures that all members of staff are on the same page.

Application Area and Required Data

This Utility Management Database is a tool for tracking city utility maintenance schedules (related to the electrical grid).

The data contained within this database is as follows:

- UTILITY_BRANCH
  - ub_id
  - name
  - ub_address_id
- ADDRESS
  - address_id
  - street
  - city
  - state
  - zip_code
  - phone_number
- WORKER
  - worker_id
  - worker_ub_id
  - worker_address_id
  - fname
  - lname
  - wage
- SCHEDULE
  - schedule_id
  - schedule_worker_id
  - start_date
  - end_date
- TASK
  - task_id
  - task_utility_id
  - task_schedule_id
  - appointment_date
- UTILITY
  - utility_id
  - description
  - longitude
  - latitude
- EQUIPMENT
  - equipment_id
  - equipment_ub_id
  - count
  - description
- EQUIPMENT_REQUIRED
  - equipment_required_task_id

        - equipment_required_equipment_id
        - count
    - EQUIPMENT_OUT
        - equipment_out_ub_id
        - equipment_out_equipment_id
        - equipment_out_worker_id
        - count

Testing Data, Generation and Methodology

Test data methodology

Wanted multiple utility branches with a variety of related records from each other entity. For example, some utility branches will have no employees, 1 employee, or many employees. By extension, some utilities will be associated with 0, 1, or many tasks. This logic applies to all entities and their related entities. We also tested inserts, updates, and deletes that would violate our triggers and constraints to confirm that they were working as intended.

Generating Test Data

We generated test data by hand to create an adequately sized database. Once we had valid data, we then tried operations that violated the databases constraints. For our TASK_SCHEDULE table, we tried inserting and updating records that had start dates after end dates, the start and end date are not two weeks apart, and that one worker will not have overlapping schedules. For our UTILITY entity, we tried inserting and updating utilities, giving them invalid latitude and longitude coordinates. For our TASK table, we tried inserting and updating records with task appointment times that do not fit within their associated schedules and tasks that are at the same exact time for the same worker. Finally, for our EQUIPMENT_OUT table, we attempted to insert and update records that would checkout more equipment than was at the utility branch, or if workers attempted to check out equipment from a branch they do not belong to. We confirmed that these test cases and triggers worked by checking that the appropriate trigger warning was displayed with the related illegal operation.

Why we commented triggers

Apparently, Azure uses ClearDB and unless the server is paid for, SUPER privileges cannot be assigned, meaning we cannot execute triggers on the server without a paid subscription.

https://getsatisfaction.com/cleardb/topics/create_function_without_super_privileges

See top response.

Changes from Iteration I to Iteration II

Utility Branch

Specified foreign key to Address.

Location/Address

Renamed Location to Address.

Gave Address a primary key.

Removed country and added zip code and phone number.

Worker

Specified foreign key to Utility Branch and Address.

Gave Worker first_name, last_name, and wage.

Schedule

Added primary key to Schedule.

Specified foreign key to Worker.

Added start date and end date for Schedule.

Light Post/Transformer/Utility

Just made a general Utility entity instead of specific.

We did this so tasks could reference utility records and identify what utility it is.

Added longitude and latitude as attributes of Utility.

Added description to Utility to specify what type of Utility the record is.

Dropped status attribute, assuming if there is a Task referencing it, it is broken.

Vehicle

Completely dropped this entity because this entity does not add any value to the project.

Equipment

Specified foreign key to Utility Branch.

Gave it a description and count attribute.

Equipment_Required

Added this entity so we can specify what equipment is required for a certain task and how much of the referenced equipment is required.

Equipment_Out

Added this table to keep track of which workers have how many of a certain piece of equipment checked out.

Gave it a foreign key to Utility Branch so we could quickly look up what equipment is checked out at a certain branch.

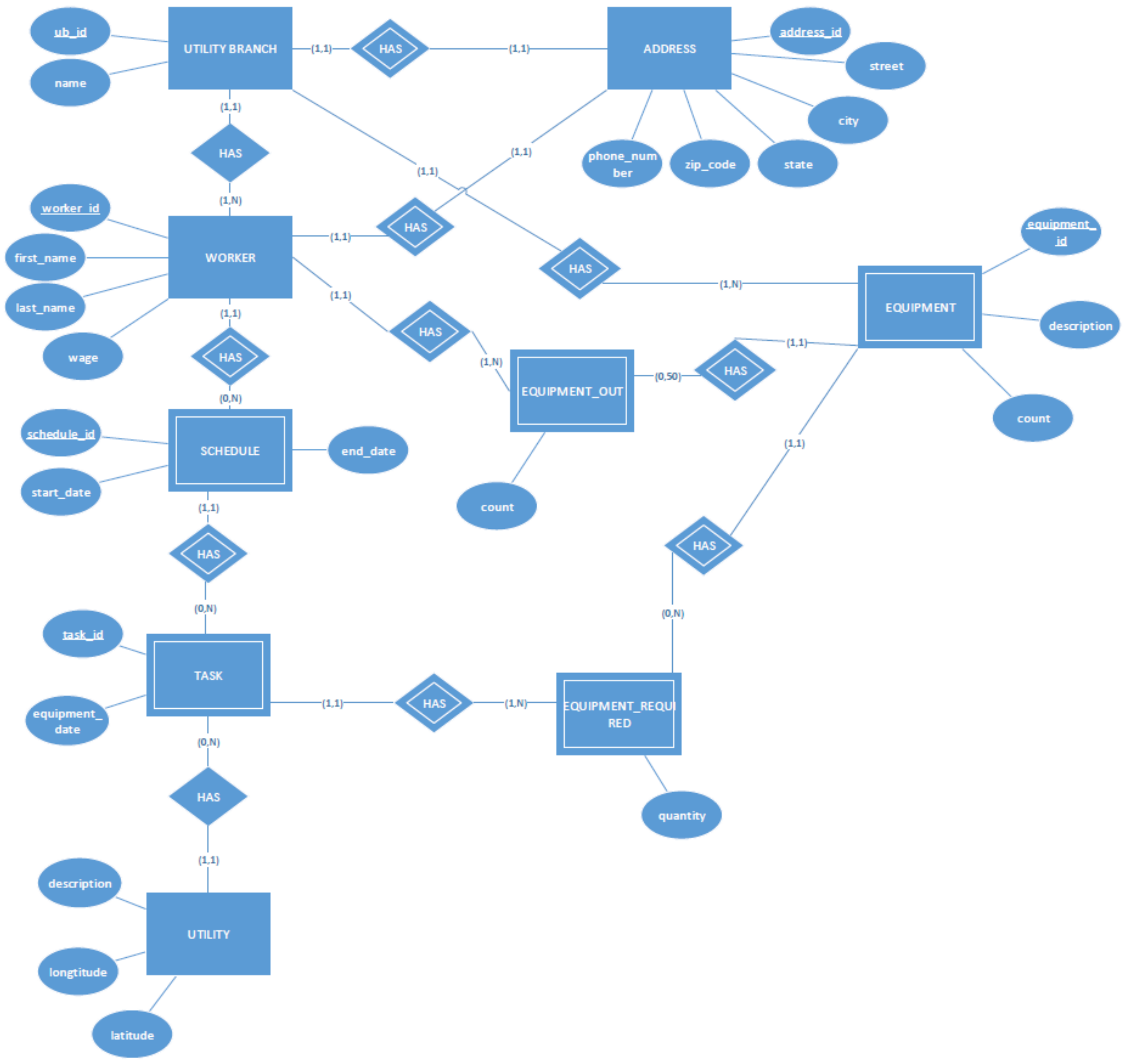
Changes from Iteration II to Iteration III

The only change made from the second to the third iteration was to the “Equipment_Out” table. For this particular case, we added “equipment_out_worker_id”, which is a foreign key to “worker_id”, to be part of the “equipment_out” primary key. This way, we were able to keep track of what workers had what equipment from particular utility branches.

Database design and results

This database application essentially keeps track of workers at utility branches, their schedules, transformers and light posts, and the tasks necessary to fix transformers and lightposts. For this narrow application, the database works. We designed with this constraint in mind and did our best to make things normalized and easily referenceable. To make things easily referenceable, we gave many entities their own primary keys. This improves efficiency of look up/retrieval, and compared to alternatives saves memory space. We also made an Address table so we could avoid multi-valued attributes in tables that would have an Address. These decisions put all relations into at least second Normal Form. In our database design, we designed our 1:N relations so that the “N” side records hold the foreign key referencing the “1”. This an example where the utility branch would have a multi-valued attribute of its workers, or there would have to be a unique utility branch record for each worker record. Figure 1 can be reference to get a visual representation of the database design, as it is our ER Diagram. Figure 2 can also be referenced as a good resource for visualizing the various dependencies and foreign keys within the database, as it is our Relational Model.

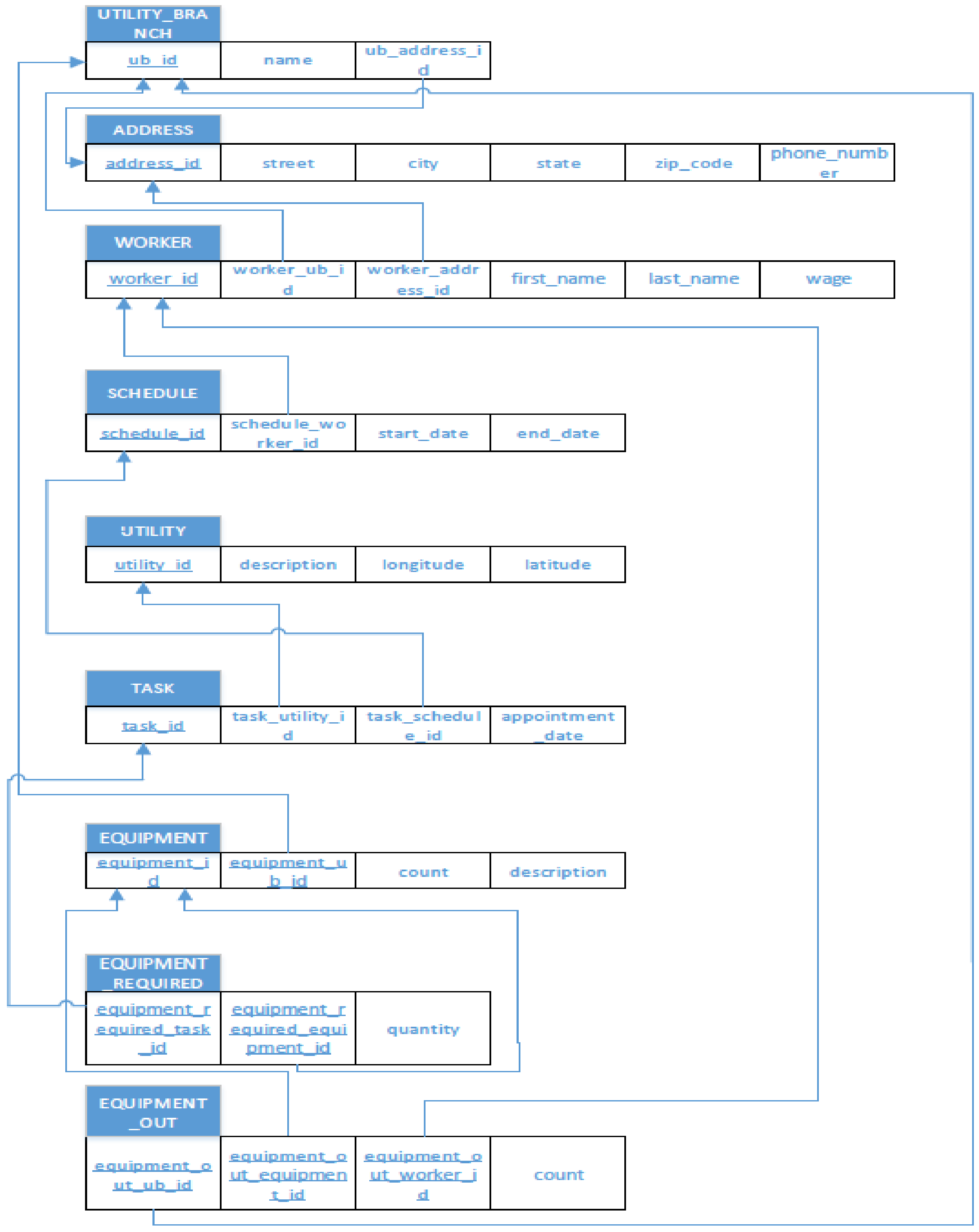
Figure 1. Entity-Relationship Diagram



Our ER Diagram (above) depicts the various relationships between the entities within our database. This includes the cardinalities of relationships, identifying types, and the various attributes associated with each individual entity.

*We would recommend you zoom in here.

Figure 2. Relational Model



Our Relational Model (above) depicts the multiple foreign key relationships between entities, and clearly shows the foreign keys contained within each table. It also allows for a quick understanding of the functional dependencies within entities.

*We would recommend you also zoom in here.

Assumptions

We made a few assumptions over the course of our database design process. First off, we assumed that tasks cannot overlap in terms of their appointment date and time. However in this application, they could be 1 second apart which is unreasonable. Because of this, we assume that the tasks will not be generated this close together, and if they are, the tasks will not be part of the same schedule. We also assumed that the user of the database will keep track of counts of equipment and equipment checked out, so that the amount of equipment that can be checked out is restricted by the amount available. This was going to be a part of our application logic but we did not have enough time to implement this.

Evaluation of project

How much effort was spent overall?

A considerable amount of effort went into this project. We always adhered to our scheduled times and met additionally when necessary, which was often. We estimate that we worked ~100 person hours.

What went right?

We did well with a number of our triggers. We were able to enforce data restrictions on Schedules and their associated Tasks. Also, our console application worked in all but one case described below. With our console application, the user is able to write their own SQL queries as well as press buttons to show them different views of the database (such as Workers and their associated Tasks). We think our design is mostly correct, with the exception of some minor normalization sacrificed for efficiency in referencing.

What went wrong?

We tried to use an Azure server for our database, which was going fine until we tried to write triggers. Free subscriptions to Azure do not allow SUPER() privileges, so we could not write triggers with the Azure database. Also, in the middle of Iteration II, we had a hard time designing the relations between workers, schedules, tasks, and utilities. At one point we tried making a specific Lightpost entity and Lightpost_Task entity, but we later determined this was a poor design and not extendable. As a result, we ended up making one Utility table and one Task table. Additionally, not all of our tables are perfectly normalized. This is largely a result of the fact that many of our entities are closely related and utilize foreign keys. With these foreign keys, we wanted records to be able to easily access records in related entities, so we often ascribed primary keys to entities. Finally, we did not fully finalize our application side of this project. Ideally, we wanted to implement Equipment count logic and Equipment_Out logic with our application.

What would we do differently if we were to do it again?

We would not use MySQL for this project. With all the issues with triggers, checks, and constraints MySQL proved frustrating. It could be interesting and useful to research other database options, such as NoSQL or PostgreSQL. Also, with the knowledge we now know about normalization, it would have been easier to design our database if we started with normalization earlier in the design process.

Given more time, what would we improve?

If we had another week, we would finish our application logic for keeping count of Equipment and Equipment_Out. Also given more time, I think we would size up the scope of our database a bit. It would be interesting to add something like financial data to this database with things like budget, wages, costs of tasks, etc. Also we would add things like departments, managers, larger projects, etc. Again, normalization would be a focus going forward. In addition, we would bring our Equipment table into Boyce-Codd normal form by splitting it up. An example of how we would do this can be seen in Figure 3.

NFs of each of our relations

Utility Branch
Yes 1NF – No multivalued attributes.
Yes 2NF – Composite primary key, so all non-prime attributes are dependent on the whole primary key.
Yes 3NF – No transitive dependencies.
Yes BCNF – In every functional dependency, the determinant is a candidate key.

Address
Yes 1NF – No multivalued attributes.
Yes 2NF – All non-prime attributes dependent on primary key, address_id.
Yes 3NF – No transitive dependencies, because all attributes are artificially dependent on primary key, address_id. *We created this primary key to make it easier to reference address from other tables.
BCNF – Probably not due to above reason.

Worker
Yes 1NF – No multivalued attributes.
Yes 2NF – All non-prime attributes are dependent on primary key, worker_id.
Yes 3NF – No transitive dependencies.
Yes BCNF – In every functional dependency, the determinant is a candidate key.

Task Schedule
Yes 1NF – No multivalued attributes.
Yes 2NF – All non-prime attributes are dependent on primary key, task_schedule_id.
Yes 3NF – No transitive dependencies.
Yes BCNF – Kind of unclear. We determined that a worker is dependent on a schedule. A worker's actions are determined based on the assigned schedule. A schedule can arbitrarily be assigned to any worker.

Utility
Yes 1NF – No multivalued attributes.
Yes 2NF – All non-prime attributes are dependent on primary key, utility_id. Latitude and longitude are not granular enough to specify a utility.
Yes 3NF – No transitive dependencies.
Yes BCNF – In every functional dependency, the determinant is a candidate key.

Task
Yes 1NF – No multivalued attributes.
Yes 2NF – All non-prime attributes are not partially dependent on primary key.
Yes 3NF – No transitive dependencies.
Yes BCNF – The only functional dependency is appointment_date is dependent on task_id, which is a candidate key.

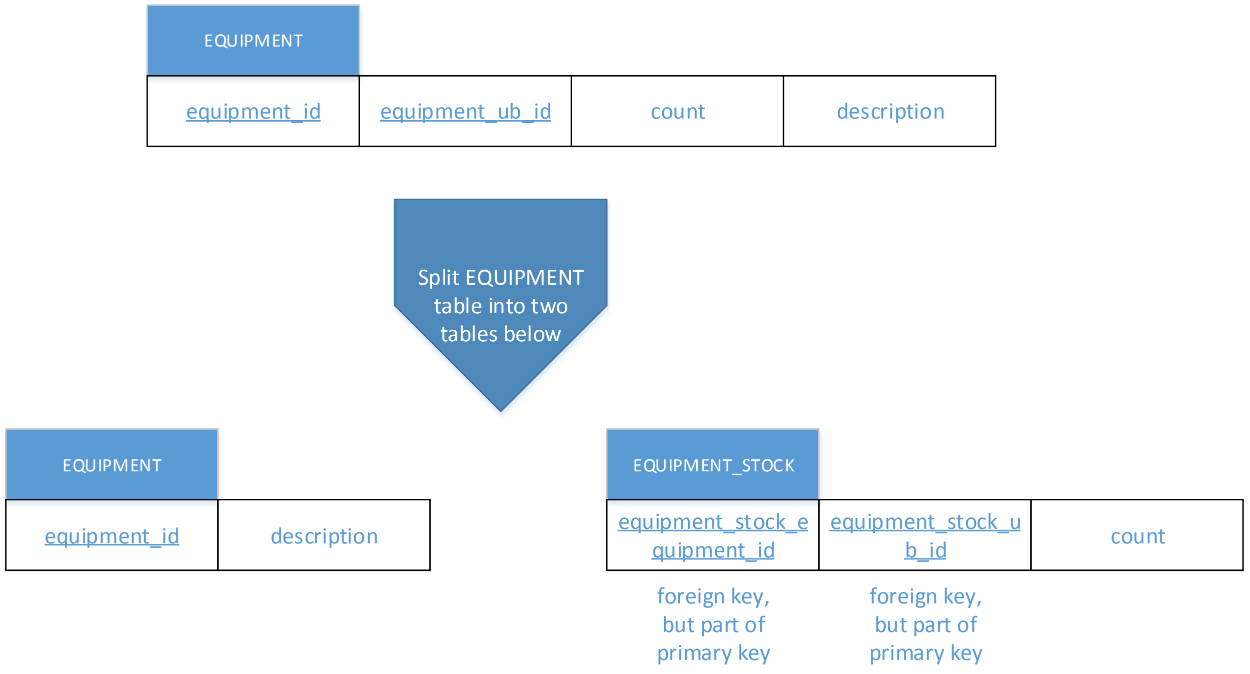
Equipment
Yes 1NF – No multivalued attributes.
Yes 2NF – Description is only dependent on equipment_id, not utility_branch_id, aka partially dependent.
No 3NF – See above.

No BCNF – See above.

Equipment_Required
Yes 1NF – No multivalued attributes.
Yes 2NF – All non-prime attributes are not partially dependent on primary keys.
Yes 3NF – No transitive dependencies.
Yes BCNF – All functional dependency determinants are candidate keys.

Equipment_Out
Yes 1NF – No multivalued attributes.
Yes 2NF – All non-prime attributes are not partially dependent on primary keys.
Yes 3NF – No transitive dependencies.
Yes BCNF – All functional dependency determinants are candidate keys.

Figure 3. Normalization Changes to our Entities



EQUIPMENT
equipment_id
equipment_ub_id
count
description
Split EQUIPMENT table into two tables below
EQUIPMENT
equipment_id
description
EQUIPMENT_STOCK
equipment_stock_equipment_id
equipment_stock_ub_id
count
foreign key, but part of primary key
foreign key, but part of primary key

We came up with multiple SQL statements that we thought would be applicable to real-world scenarios. The table below documents the syntax used to accomplish these desired queries along with the reason for their existence. Later on, these exact queries are implemented into our application side of the database, which is document in the console application portion of this document.

SQL Statements and their Purpose

SQL Statement	Purpose

select worker.last_name, worker.worker_id, worker.worker_utility_branch_id as utility_branch, equipment.description, equipment_out.count
from equipment, worker
join equipment_out
on
worker.worker_id = equipment_out.equipment_out_worker_id
where equipment_out.equipment_out_equipment_id = equipment.equipment_id;
	Used to view all workers with equipment checked out, and to see what specific equipment and how much of that equipment each worker has.

select w.worker_utility_branch_id as utility_branch, w.worker_id, address.street, address.city, address.state, address.zip_code as zip, address.phone_number as phone
from worker w
join address
on
w.worker_address_id = address.address_id
order by w.worker_utility_branch_id;
	Used to view the contact information of every single worker, and is ordered by the workers' utility branch id so that it is easy to navigate between employees at specific branches.

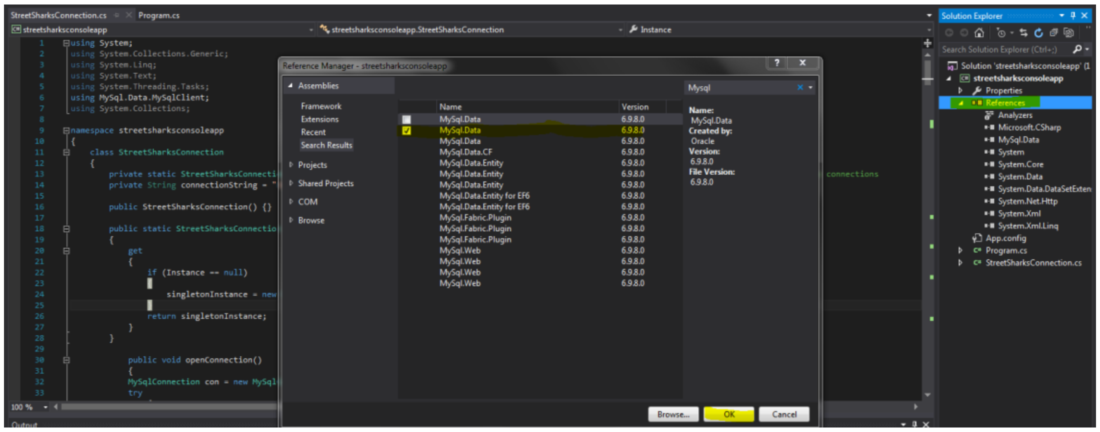
select utility.*, task.task_id, task.appointment_date from utility left join task on utility.utility_id = task.task_utility_id;	Performs a left join on Utility with Task to view every single utility and if they have an associated task or not. Tells the user what utilities are broken and which ones aren't, along with what utilities have associated tasks and are therefore part of worker schedules.
select u.description, u.longitude, u.latitude, w.worker_id, w.last_name, t.appointment_date from utility u, worker w, task t, task_schedule s where DATE(t.appointment_date) = CURRENT_DATE and s.schedule_worker_id = w.worker_id and s.schedule_id = t.task_schedule_id and t.task_utility_id = u.utility_id;	Shows what tasks are assigned for today, so users can determine what work is essential to be completed by the end of the workday. Also reminds workers of what their priorities are for the day.
select u.description, u.utility_id, u.latitude, u.longitude, e.description as equip_req_desc, er.quantity as amt_needed	Used to display only utilities that have tasks assigned to them, along with the equipment that is required to complete that tasks.

from utility u, task t, equipment e, equipment_required er where u.utility_id = t.task_utility_id and er.equipment_required_task_id = t.task_id and er.equipment_required_equipment_id = e.equipment_id;	Included the description of the equipment instead of just the id, so that users can read what the equipment is in English.
select w.first_name, w.last_name, u.description, t.task_id, t.appointment_date from worker w, utility u, task t, task_schedule s where s.schedule_worker_id = w.worker_id and t.task_schedule_id = s.schedule_id and u.utility_id = t.task_utility_id;	Shows a list of all workers and tasks that are assigned to them, along with descriptive information as to what the utility is that they will be working on and the time of the appointment, so they can get a quick view of their schedule.
select u.utility_branch_name, u.utility_branch_id, e.description, e.quantity, eo.count as quantityOut from equipment e, equipment_out eo, utility_branch u where u.utility_branch_id = e.equipment_utility_branch_id and eo.equipment_out_equipment_id = e.equipment_id;	Shows all utility branches and the tools that they own. It displays the amount of each tool that actually belongs to them, but aren't necessarily in stock. The amount out column shows how many of the tools of a specific type for each utility branch are checked out, and therefore are not at the utility branch.
SHOW TABLES;	Shows a list of all tables so users can be familiarized with the structure of the database, along with naming conventions for custom queries.
SHOW FIELDS FROM " + tblName;	Shows all attributes and their specified properties for a specified table, which can be used for naming conventions and custom queries. Also helps users become familiarized with the database.

- NOTE: We also have an option for custom queries, where the user can enter a query of their desired choice if it is not one of the queries offered by default. It catches errors with naming conventions and syntax. This enables the console application to be completely customizable for any user needs beyond the ones offered.

Related to the table above, we now explain the use of our application side of the database which utilizes the exact queries specified.

Console Application Usage

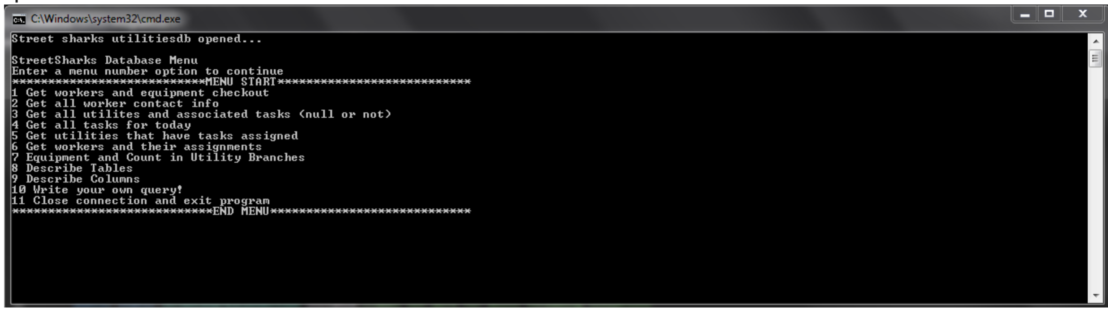
1. Make sure you have at least Visual Studio 2015 and are running a Windows 7 or greater operating system. before trying to deploy the Street Sharks Utilities Database Console Application
2. Download MYSQL Connector/NET
    a. First go to this link: https://dev.mysql.com/downloads/connector/net/
    b. Download the correct MSI for your Windows Configuration
3. After downloading, open up visual studio and create a new C# Console Application project
    a. Add two classes, StreetSharksConnection.cs, and Program.cs
4. Copy and paste the code provided into the correct class (Street Sharks Connection is the long one)
5. On the right side of Visual Studio, in the Solution Explorer, right click on References and click Add Reference
    a. In the Search Assemblies search bar, search MySql and select mysql.data, then click ok, as shown below
    b. 

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6   using MySql.Data.MySqlClient;
7   using System.Collections;
8
9   namespace streetsharksconsoleapp
10  {
11      class StreetSharksConnection
12      {
13          private static StreetSharksConnecti
14          private String connectionString = "
15
16          public StreetSharksConnection() {}
17
18          public static StreetSharksConnectio
19          {
20              get
21              {
22                  if (Instance == null)
23
24                      singletonInstance = new
25
26                  return singletonInstance;
27              }
28          }
29
30              public void openConnection()
31              {
32              MySqlConnection con = new MySql
33              try
```

	Name	Version
☐	MySql.Data	6.9.8.0
✓	MySql.Data	6.9.8.0
	MySql.Data	6.9.8.0
	MySql.Data.CF	6.9.8.0
	MySql.Data.Entity	6.9.8.0
	MySql.Data.Entity	6.9.8.0
	MySql.Data.Entity	6.9.8.0
	MySql.Data.Entity for EF6	6.9.8.0
	MySql.Data.Entity for EF6	6.9.8.0
	MySql.Fabric.Plugin	6.9.8.0
	MySql.Fabric.Plugin	6.9.8.0
	MySql.Fabric.Plugin	6.9.8.0
	MySql.Web	6.9.8.0
	MySql.Web	6.9.8.0
	MySql.Web	6.9.8.0
	MySql.Web	6.9.8.0

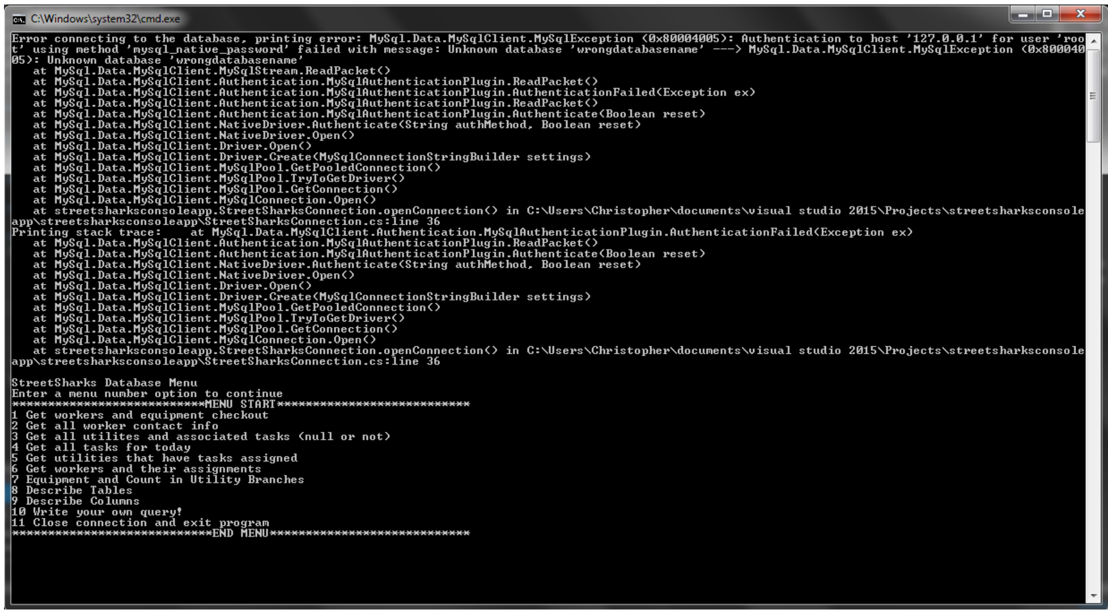
Name:
MySql.Data
Created by:
Oracle
Version:
6.9.8.0
File Version:
6.9.8.0

6. Create the database in MySQL workbench using CSS475 class directions
7. Now, in the Console Application, select the StreetSharksConnection.cs file
8. On line 14, where the connectionString instance variable is declared, change the code shown below with your particular inserts, for example, where it says [insert_db_name_here_no_braces] type in the database name without using the square brackets, and the same for the insert localhost, user id and password.
    a. private String connectionString = "Database=[insert_db_name_here_no_braces];Data Source=[insert_localhost_or_127.0.0.1_or_server_here_no_braces];User Id=[insert_user_id_here_no_braces];Password=[insert_password_here_no_braces]";

9. The project should be built and be compiled by clicking run -> the connection should open and will open in a Console window shown below
   a. 

```
Street sharks utilitiesdb opened...

StreetSharks Database Menu
Enter a menu number option to continue
*************************MENU START***************************
1 Get workers and equipment checkout
2 Get all worker contact info
3 Get all utilites and associated tasks (null or not)
4 Get all tasks for today
5 Get utilities that have tasks assigned
6 Get workers and their assignments
7 Equipment and Count in Utility Branches
8 Describe Tables
9 Describe Columns
10 Write your own query!
11 Close connection and exit program
****************************END MENU****************************
```

   b. If the database was unable to be connected, an exception will be thrown and the output below will be shown, this means that the connectionString was not input correctly
   c. 

```
Error connecting to the database, printing error: MySql.Data.MySqlClient.MySqlException (0x80004005): Authentication to host '127.0.0.1' for user 'roo
t' using method 'mysql_native_password' failed with message: Unknown database 'wrongdatabasename' ---> MySql.Data.MySqlClient.MySqlException (0x800040
05): Unknown database 'wrongdatabasename'
   at MySql.Data.MySqlClient.MySqlStream.ReadPacket()
   at MySql.Data.MySqlClient.Authentication.MySqlAuthenticationPlugin.ReadPacket()
   at MySql.Data.MySqlClient.Authentication.MySqlAuthenticationPlugin.AuthenticationFailed(Exception ex)
   at MySql.Data.MySqlClient.Authentication.MySqlAuthenticationPlugin.ReadPacket()
   at MySql.Data.MySqlClient.Authentication.MySqlAuthenticationPlugin.Authenticate(Boolean reset)
   at MySql.Data.MySqlClient.NativeDriver.Authenticate(String authMethod, Boolean reset)
   at MySql.Data.MySqlClient.NativeDriver.Open()
   at MySql.Data.MySqlClient.Driver.Open()
   at MySql.Data.MySqlClient.Driver.Create(MySqlConnectionStringBuilder settings)
   at MySql.Data.MySqlClient.MySqlPool.GetPooledConnection()
   at MySql.Data.MySqlClient.MySqlPool.TryToGetDriver()
   at MySql.Data.MySqlClient.MySqlPool.GetConnection()
   at MySql.Data.MySqlClient.MySqlConnection.Open()
   at streetsharksconsoleapp.StreetSharksConnection.openConnection() in C:\Users\Christopher\documents\visual studio 2015\Projects\streetsharksconsole
app\streetsharksconsoleapp\StreetSharksConnection.cs:line 36
Printing stack trace:    at MySql.Data.MySqlClient.Authentication.MySqlAuthenticationPlugin.AuthenticationFailed(Exception ex)
   at MySql.Data.MySqlClient.Authentication.MySqlAuthenticationPlugin.ReadPacket()
   at MySql.Data.MySqlClient.Authentication.MySqlAuthenticationPlugin.Authenticate(Boolean reset)
   at MySql.Data.MySqlClient.NativeDriver.Authenticate(String authMethod, Boolean reset)
   at MySql.Data.MySqlClient.NativeDriver.Open()
   at MySql.Data.MySqlClient.Driver.Open()
   at MySql.Data.MySqlClient.Driver.Create(MySqlConnectionStringBuilder settings)
   at MySql.Data.MySqlClient.MySqlPool.GetPooledConnection()
   at MySql.Data.MySqlClient.MySqlPool.TryToGetDriver()
   at MySql.Data.MySqlClient.MySqlPool.GetConnection()
   at MySql.Data.MySqlClient.MySqlConnection.Open()
   at streetsharksconsoleapp.StreetSharksConnection.openConnection() in C:\Users\Christopher\documents\visual studio 2015\Projects\streetsharksconsole
app\streetsharksconsoleapp\StreetSharksConnection.cs:line 36

StreetSharks Database Menu
Enter a menu number option to continue
*************************MENU START***************************
1 Get workers and equipment checkout
2 Get all worker contact info
3 Get all utilites and associated tasks (null or not)
4 Get all tasks for today
5 Get utilities that have tasks assigned
6 Get workers and their assignments
7 Equipment and Count in Utility Branches
8 Describe Tables
9 Describe Columns
10 Write your own query!
11 Close connection and exit program
****************************END MENU****************************
```

Now that an understanding of the use of our database has been given, our script for generating the table and inserting various pieces of data for testing and verification of implemented constraints.

Final SQL Script

```
-- this is a table that holds utility branches
-- utility branches will store workers and utilities
-- found in the city and associated with that utility branch
CREATE TABLE UTILITY_BRANCH
(
utility_branch_id DECIMAL(9,0) UNSIGNED NOT NULL,
utility_branch_name VARCHAR(20) NOT NULL UNIQUE,
utility_branch_address_id DECIMAL(9,0) UNSIGNED NOT NULL,
PRIMARY KEY(utility_branch_id)
);

-- address is a holder table that stores everything
-- needed to know for an address (anything can point
-- to an address, a worker, a utility branch, and anything
-- extended to this database
CREATE TABLE ADDRESS
(
address_id DECIMAL(9,0) UNSIGNED NOT NULL,
street VARCHAR(100) NOT NULL,
city VARCHAR(40) NOT NULL,
state CHAR(2) NOT NULL,
zip_code DECIMAL(5,0) UNSIGNED NOT NULL,
phone_number DECIMAL(10,0) UNSIGNED NOT NULL,
PRIMARY KEY(address_id)
);

-- this table holds worker values
-- which will point to a certain utility_branch
-- and the values required for a worker to hold,
-- such as wages and their address, tasks and schedules
-- will point to worker so that they can reference
-- who is doing the job and when
CREATE TABLE WORKER
(
worker_id DECIMAL(9,0) UNSIGNED NOT NULL,
worker_utility_branch_id DECIMAL(9,0) UNSIGNED NOT NULL,
first_name VARCHAR(25) NOT NULL,
last_name VARCHAR(25) NOT NULL,
worker_address_id DECIMAL(9,0) UNSIGNED NOT NULL,
wage DECIMAL(7, 2) UNSIGNED NOT NULL,
PRIMARY KEY(worker_id)
);
```

```
-- this table is essentially the schedule for a worker
-- although schedule is a pre-named name in mysql
-- this schedule is assumed to be a 2 week period
-- associated with a worker
CREATE TABLE TASK_SCHEDULE
(
schedule_id DECIMAL(9,0) UNSIGNED NOT NULL,
schedule_worker_id DECIMAL(9,0) UNSIGNED NOT NULL,
start_date DATE NOT NULL,
end_date DATE NOT NULL,
PRIMARY KEY(schedule_id)
);

-- a trigger that prevents schedule dates from overlapping on a worker and also prevents
overlapping schedules on insert
DELIMITER $$
CREATE TRIGGER preventInvalidScheduleDates BEFORE INSERT ON TASK_SCHEDULE
FOR EACH ROW
BEGIN
			IF ((NEW.start_date > NEW.end_date) OR (NEW.start_date > (NEW.end_date -
INTERVAL 2 WEEK))) THEN
		SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FAILURE: START DATE CANNOT BE
AFTER END DATE OR LESS THAN A 2 WEEK PERIOD';
	END IF;
		IF EXISTS(SELECT ts.schedule_worker_id FROM TASK_SCHEDULE ts WHERE
ts.schedule_worker_id = NEW.schedule_worker_id AND
			((NEW.start_date >= ts.start_date AND
			NEW.start_date <= ts.end_date) OR
			(NEW.end_date <= ts.end_date AND
			NEW.end_date >= ts.start_date))) THEN
			SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FAILURE: ONE WORKER CANT
HAVE OVERLAPPING SCHEDULES';
	END IF;
END$$
DELIMITER ;

-- a trigger that prevents schedule dates from overlapping on a worker and also prevents
overlapping schedules on update
DELIMITER $$
CREATE TRIGGER preventInvalidScheduleDates2 BEFORE UPDATE ON TASK_SCHEDULE
FOR EACH ROW
BEGIN
```

```
                    IF ((NEW.start_date > NEW.end_date) OR (NEW.start_date > (NEW.end_date -
INTERVAL 2 WEEK))) THEN
             SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FAILURE: START DATE CANNOT BE
AFTER END DATE OR LESS THAN A 2 WEEK PERIOD';
      END IF;
             IF EXISTS(SELECT ts.schedule_worker_id FROM TASK_SCHEDULE ts WHERE
ts.schedule_worker_id = NEW.schedule_worker_id AND
                    ((NEW.start_date >= ts.start_date AND
                    NEW.start_date <= ts.end_date) OR
                    (NEW.end_date <= ts.end_date AND
                    NEW.end_date >= ts.start_date))) THEN
                    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FAILURE: ONE WORKER CANT
HAVE OVERLAPPING SCHEDULES';
      END IF;
END$$
DELIMITER ;

-- a table that stores different types of utilities that a city would manage
-- in our case we only added LP and TF which stood for LightPost and Transformer
-- as well as the global positioning coordinates for said utility
CREATE TABLE UTILITY
(
utility_id DECIMAL(9,0) UNSIGNED NOT NULL,
description CHAR(2) NOT NULL,
latitude DECIMAL(6, 4) NOT NULL UNIQUE,
longitude DECIMAL(7, 4) NOT NULL UNIQUE,
PRIMARY KEY(utility_id)
);

-- trigger to prevent invalid coordinates from being entered on insert
DELIMITER $$
CREATE TRIGGER preventInvalidCoordinates BEFORE INSERT ON UTILITY
FOR EACH ROW
BEGIN
                              IF NEW.latitude > 90 OR NEW.latitude < -90 THEN
         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FAILURE: INVALID COORDINATES';
         END IF;
         IF NEW.longitude > 180 OR NEW.longitude < -180 THEN
                      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FAILURE: INVALID
COORDINATES';
         END IF;
END$$
DELIMITER ;
```

```
-- trigger to prevent invalid coordinates from being entered on update
DELIMITER $$
CREATE TRIGGER preventInvalidCoordinates2 BEFORE UPDATE ON UTILITY
FOR EACH ROW
BEGIN
                IF NEW.latitude > 90 OR NEW.latitude < -90 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FAILURE: INVALID COORDINATES';
    END IF;
    IF NEW.longitude > 180 OR NEW.longitude < -180 THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FAILURE: INVALID
COORDINATES';
    END IF;
END$$
DELIMITER ;

-- task table is a table that is used to piece together a worker
-- with a schedule, and fill up a worker's schedule with various tasks
-- at various times,
-- it is the crux of the database, as it stores a worker reference, a
-- schedule references and a utility reference
-- as well as the exact time that the appointment is scheduled for
-- we assume that appointment dates and times would be entered correctly,
-- that is, a worker could work on one task and then the next minute,
-- be working on another
CREATE TABLE TASK
(
task_id DECIMAL(9,0) UNSIGNED NOT NULL,
task_utility_id DECIMAL(9,0) UNSIGNED NOT NULL,
task_schedule_id DECIMAL(9,0) UNSIGNED NOT NULL,
appointment_date DATETIME,
PRIMARY KEY(task_id)
);

-- a trigger to prevent an appointment date from being before the schedule beginning
-- or after the schedule end, as well as preventing from appointment dates being
-- at exactly the same time (over lapping) on insert
DELIMITER $$
CREATE TRIGGER preventInvalidApptDate BEFORE INSERT ON TASK
FOR EACH ROW
BEGIN
                -- task must be within schedule time
                IF NEW.appointment_date < (select ts.start_date from task_schedule ts
where NEW.task_schedule_id = ts.schedule_id) THEN
```

```
	SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FAILURE: APPOINTMENT DATE CANNOT
BE BEFORE BEGINNING OF SCHEDULE';
	END IF;
	IF NEW.appointment_date > (select ts.end_date from task_schedule ts where
NEW.task_schedule_id = ts.schedule_id) THEN
	SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FAILURE: APPOINTMENT DATE CANNOT
BE AFTER END OF SCHEDULE';
	END IF;
				-- checks if there is already an appointment date or not
				IF NEW.appointment_date = ANY(select t.appointment_date from
task_schedule ts, task t where NEW.task_schedule_id = ts.schedule_id) THEN
			SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FAILURE: APPOINTMENT
DATE CANNOT BE AT THE SAME TIME AS AN EXISTING APPOINTMENT';
				END IF;
END$$
DELIMITER ;

-- a trigger to prevent an appointment date from being before the schedule beginning
-- or after the schedule end, as well as preventing from appointment dates being
-- at exactly the same time (over lapping) on update
DELIMITER $$
CREATE TRIGGER preventInvalidApptDate2 BEFORE UPDATE ON TASK
FOR EACH ROW
BEGIN
				-- task must be within schedule time
				IF NEW.appointment_date < (select ts.start_date from task_schedule ts
where NEW.task_schedule_id = ts.schedule_id) THEN
	SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FAILURE: APPOINTMENT DATE CANNOT
BE BEFORE BEGINNING OF SCHEDULE';
	END IF;
	IF NEW.appointment_date > (select ts.end_date from task_schedule ts where
NEW.task_schedule_id = ts.schedule_id) THEN
	SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FAILURE: APPOINTMENT DATE CANNOT
BE AFTER END OF SCHEDULE';
	END IF;
				-- checks if there is already an appointment date or not
				IF NEW.appointment_date = ANY(select t.appointment_date from
task_schedule ts, task t where NEW.task_schedule_id = ts.schedule_id) THEN
			SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FAILURE: APPOINTMENT
DATE CANNOT BE AT THE SAME TIME AS AN EXISTING APPOINTMENT';
				END IF;
END$$
DELIMITER ;
```

```
-- equipment table is used to store various equipment
-- used for certain tasks and stored at various utility branches
CREATE TABLE EQUIPMENT
(
equipment_id DECIMAL(9,0) UNSIGNED NOT NULL,
equipment_utility_branch_id DECIMAL(9,0) UNSIGNED NOT NULL,
quantity DECIMAL(6,0) NOT NULL,
description VARCHAR(250) NOT NULL,
PRIMARY KEY(equipment_id, equipment_utility_branch_id)
);

-- equipment required table will store all of the certain equipment required
-- associated with completing a task as well as the quantity
-- and will point to various tasks and equipments as necessary
CREATE TABLE EQUIPMENT_REQUIRED
(
equipment_required_task_id DECIMAL(9,0) UNSIGNED NOT NULL,
equipment_required_equipment_id DECIMAL(9,0) UNSIGNED NOT NULL,
quantity DECIMAL(2,0) NOT NULL,
PRIMARY KEY(equipment_required_task_id, equipment_required_equipment_id)
);

-- this table is used to keep track of how much equipment is currently
-- out at a certain branch, as well as what worker has what equipment
-- we make an assumption on this table that we didn't do in our console app
-- because the project is more about databasing and less about console app
-- we did NOT automatically update the count in equipment when an
-- equipment_out record is entered
CREATE TABLE EQUIPMENT_OUT
(
equipment_out_equipment_id DECIMAL(9,0) UNSIGNED NOT NULL,
equipment_out_utility_branch_id DECIMAL(9,0) UNSIGNED NOT NULL,
equipment_out_worker_id DECIMAL(9,0) UNSIGNED NOT NULL,
count DECIMAL(6,0) NOT NULL,
PRIMARY KEY(equipment_out_equipment_id, equipment_out_utility_branch_id,
equipment_out_worker_id)
);

-- a trigger to prevent a user from pulling out insufficient stock from an equipment in a certain
utility branch on insert
-- we make an assumption on this table that we didn't do in our console app
-- because the project is more about databasing and less about console app
-- we did NOT automatically update the count in equipment when an
-- equipment_out record is entered
```

```
DELIMITER $$
CREATE TRIGGER preventOverCheckOut BEFORE INSERT ON EQUIPMENT_OUT
FOR EACH ROW
BEGIN
                                IF NEW.count > ANY (select quantity from equipment where
equipment_utility_branch_id = NEW.equipment_out_utility_branch_id) THEN
          SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FAILURE: INSUFFICIENT STOCK TO
CHECK OUT THAT ITEM';
          END IF;
          IF NEW.equipment_out_utility_branch_id != ANY (select worker_utility_branch_id from
worker where worker_id = new.equipment_out_worker_id) THEN
          SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FAILURE: WORKER CANNOT CHECK OUT
FROM OTHER BRANCHES';
          END IF;
END$$
DELIMITER ;

-- a trigger to prevent a user from pulling out insufficient stock from an equipment in a certain
utility branch on update
-- we make an assumption on this table that we didn't do in our console app
-- because the project is more about databasing and less about console app
-- we did NOT automatically update the count in equipment when an
-- equipment_out record is entered
DELIMITER $$
CREATE TRIGGER preventOverCheckOut2 BEFORE UPDATE ON EQUIPMENT_OUT
FOR EACH ROW
BEGIN
                                IF NEW.count > (select quantity from equipment where
equipment_utility_branch_id = NEW.equipment_out_utility_branch_id) THEN
          SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FAILURE: INSUFFICIENT STOCK TO
CHECK OUT THAT ITEM';
          END IF;
          IF NEW.equipment_out_utility_branch_id != (select worker_utility_branch_id from
worker where worker_id = new.equipment_out_worker_id) THEN
          SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FAILURE: WORKER CANNOT CHECK OUT
FROM OTHER BRANCHES';
          END IF;
END$$
DELIMITER ;

-- simple constraint adding at the end for foreign key references on each table
-- we did all of our foreign key creations at the end of the creation of the tables because
-- this way no compilation errors will occur as going down and we don't have to
-- order our tables properly
-- start foreign key creations
ALTER TABLE UTILITY_BRANCH
ADD CONSTRAINT UBFK1
FOREIGN KEY (utility_branch_address_id) REFERENCES ADDRESS(address_id)
ON DELETE CASCADE
ON UPDATE CASCADE;

ALTER TABLE WORKER
ADD CONSTRAINT ADDRFK1
FOREIGN KEY (worker_address_id) REFERENCES ADDRESS(address_id)
ON DELETE CASCADE
ON UPDATE CASCADE,

ADD CONSTRAINT UBFK2
FOREIGN KEY(worker_utility_branch_id) REFERENCES utility_branch(utility_branch_id)
ON DELETE CASCADE
ON UPDATE CASCADE;

ALTER TABLE TASK_SCHEDULE
ADD CONSTRAINT TSFK1
FOREIGN KEY (schedule_worker_id) REFERENCES WORKER(worker_id)
ON DELETE CASCADE
ON UPDATE CASCADE;

ALTER TABLE TASK
ADD CONSTRAINT TFK1
FOREIGN KEY (task_utility_id) REFERENCES UTILITY(utility_id)
ON DELETE CASCADE
ON UPDATE CASCADE,

ADD CONSTRAINT TFK2
FOREIGN KEY (task_schedule_id) REFERENCES TASK_SCHEDULE(schedule_id)
ON DELETE CASCADE
ON UPDATE CASCADE;

ALTER TABLE EQUIPMENT
ADD CONSTRAINT EFK1
FOREIGN KEY (equipment_utility_branch_id) REFERENCES UTILITY_BRANCH(utility_branch_id)
ON DELETE CASCADE
ON UPDATE CASCADE;

ALTER TABLE EQUIPMENT_REQUIRED
ADD CONSTRAINT EQRFK1
```

```
FOREIGN KEY (equipment_required_equipment_id) REFERENCES EQUIPMENT(equipment_id)
ON DELETE CASCADE
ON UPDATE CASCADE,

ADD CONSTRAINT EQRFK2
FOREIGN KEY (equipment_required_task_id) REFERENCES TASK(task_id)
ON DELETE CASCADE
ON UPDATE CASCADE;

ALTER TABLE EQUIPMENT_OUT
ADD CONSTRAINT EQOFK1
FOREIGN KEY (equipment_out_worker_id) REFERENCES WORKER(worker_id)
ON DELETE CASCADE
ON UPDATE CASCADE,

ADD CONSTRAINT EQOFK2
FOREIGN KEY (equipment_out_equipment_id) REFERENCES EQUIPMENT(equipment_id)
ON DELETE CASCADE
ON UPDATE CASCADE,

ADD CONSTRAINT EQ0FK3
FOREIGN KEY (equipment_out_utility_branch_id) REFERENCES
UTILITY_BRANCH(utility_branch_id)
ON DELETE CASCADE
ON UPDATE CASCADE;
-- end foreign key creations

-- begin inserts
insert into address values(000000001, '1st Ave', 'Seattle', 'WA', 98121, 2065550123);
insert into address values(000000002, '2nd Ave', 'Seattle', 'WA', 98121, 2065550124); -- 002
insert into utility_branch values(000000001, 'Original', 000000001);
insert into worker values(000000001, 000000001,  'John', 'Smith', 000000002, 99999.99); -- 002
insert into task_schedule values(000000001, 00000001, '2015-11-16', '2015-11-30');
insert into utility values(000000001, 'TF', 47.6039, -122.3342);
insert into utility values(000000002, 'LP', 41.0002, -120.0002); -- 002
insert into equipment values(000000001, 000000001, 1, 'Hammer');
insert into task values(000000001, 000000001, 000000001, '2015-11-18 17:30:00');
insert into equipment_required values(000000001, 000000001, 1);
insert into equipment_out values(000000001, 000000001, 000000001, 1);
insert into equipment values(000000002, 000000001, 1, 'Wrench'); -- 002
insert into task values(000000002, 000000002, 000000001, '2015-11-18 17:41:00'); -- 002
insert into equipment_required values(000000002, 000000002, 1); -- 002
insert into equipment_out values(000000002, 000000001, 000000001, 1); -- 002
insert into address values(000000003, '3rd Ave', 'Bothell', 'WA', 98107, 2065550125);
insert into address values(000000004, '4th Ave', 'Bothell', 'WA', 98107, 2065550126); -- 004
insert into utility_branch values(000000003, 'Unoriginal', 000000003);
insert into worker values(000000003, 000000003, 'John', 'Daley', 000000004, 43000.99); -- 004
insert into utility values(000000003, 'TF', 31.6048, -89.0413);
insert into utility values(000000004, 'LP', 11.0001, -34.0784); -- 004
insert into equipment values(000000004, 000000003, 1, 'Wrench'); -- 004
insert into task values(000000004, 000000004, 000000001,'2015-11-18 16:29:00'); -- 004

insert into address values(000000005, '5th Ave', 'Monroe', 'WA', 98272, 2065550127);
insert into address values(000000006, '6th Ave', 'Monroe', 'WA', 98272, 2065550128);
insert into address values(000000007, '7th Ave', 'Monroe', 'WA', 98272, 2065550129);
insert into address values(000000008, '8th Ave', 'Monroe', 'WA', 98272, 2065550130);
insert into utility_branch values(000000005, 'UBIII', 000000005);
insert into worker values(000000005, 000000005, 'John', 'Mcmahon', 000000006, 56777.99);
insert into worker values(000000006, 000000005, 'John', 'Tate', 000000007, 21340.99);
insert into worker values(000000007, 000000005, 'John', 'Lasicky', 000000008, 87450.21);
insert into task_schedule values(000000005, 00000005, '2015-11-16', '2015-11-30');
insert into task_schedule values(000000006, 00000006, '2015-11-16', '2015-11-30');
insert into task_schedule values(000000007, 00000007, '2015-11-16', '2015-11-30');

insert into utility values(0000000010, 'TF', 64.213, -41.897);
insert into utility values(000000011, 'LP', 9.0321, -46.704);
insert into utility values(000000012, 'TF', 57.6048, -21.0413);
insert into utility values(000000013, 'LP', 13.8760, -10.0784);
insert into utility values(000000014, 'TF', 52.6048, -87.0413);
insert into utility values(000000015, 'LP', 12.3412, -10.1234);

insert into equipment values(000000006, 000000005, 1, 'Hammer');
insert into equipment values(000000007, 000000005, 1, 'Wrench');
insert into equipment values(000000008, 000000005, 1, 'Wire');
insert into task values(000000005, 0000000010, 000000005, '2015-11-18 14:30:00');
insert into equipment_required values(000000005, 000000006, 1);
insert into equipment_out values(000000008, 000000005, 000000006, 1);

insert into task_schedule values(000000009, 00000007, '2015-12-01', '2015-12-15');
insert into task values(000000006, 0000000010, 000000009, '2015-12-07 08:30:00');
insert into task values(000000007, 0000000011, 000000009, '2015-12-07 13:45:00');
insert into task values(000000008, 0000000013, 000000009, '2015-12-09 10:00:00');
insert into task values(000000009, 0000000014, 000000009, '2015-12-09 20:15:00');
insert into task values(000000010, 0000000010, 000000009, '2015-12-05 15:37:46');
insert into task values(000000011, 0000000015, 000000009, '2015-12-05 16:37:46');
-- end inserts
```

Final Console Application Source Code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using streetsharksconsoleapp;

namespace streetsharksconsoleapp
{
   class Program
   {
      static void Main(string[] args)
      {
         Execute();
      }

      private static void Execute()
      {
         StreetSharksConnection db = new StreetSharksConnection();
         db.openConnection();
         db.printMenuReadInput();
      }
   }
}



using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using MySql.Data.MySqlClient;
using System.Collections;

namespace streetsharksconsoleapp
{
   class StreetSharksConnection
   {
      private static StreetSharksConnection singletonInstance; //stops the program from having multiple
instantiatons of the same connection outside of the connections
      private String connectionString = "Database=[insert_db_name_here_no_braces];Data
Source=[insert_localhost_or_127.0.0.1_or_server_here_no_braces];User
Id=[insert_user_id_here_no_braces];Password=[insert_password_here_no_braces]";
```

```
public StreetSharksConnection() {}

public static StreetSharksConnection Instance //does not work for multi threading situations
{
   get
   {
      if (Instance == null)
      {
         singletonInstance = new StreetSharksConnection();
      }
      return singletonInstance;
   }
}

   public void openConnection()
   {
   MySqlConnection con = new MySqlConnection(connectionString);
   try
      {

         con.Open();
         Console.WriteLine("Street sharks utilitiesdb opened...");
      }
      catch (MySqlException error)
      {
         Console.WriteLine("Error connecting to the database, printing error: " + error.ToString());
         Console.WriteLine("Printing stack trace: " + error.StackTrace);
      }
      finally
      {
         if (con != null)
         {
            con.Close();
         }
     }
   }

   private void endProgram()
   {
      Console.WriteLine("Closing connection...thanks for using the streetsharks utilitiesdb!");
   }

   public void printMenuReadInput()
   {
      Console.WriteLine();
      Console.WriteLine("StreetSharks Database Menu\nEnter a menu number option to continue");
```

```
            Console.WriteLine("***************************MENU
START***************************");
            Console.WriteLine("1 Get workers and equipment checkout");
            Console.WriteLine("2 Get all worker contact info");
            Console.WriteLine("3 Get all utilites and associated tasks (null or not)");
            Console.WriteLine("4 Get all tasks for today");
            Console.WriteLine("5 Get utilities that have tasks assigned");
            Console.WriteLine("6 Get workers and their assignments");
            Console.WriteLine("7 Equipment and Count in Utility Branches");
            Console.WriteLine("8 Describe Tables");
            Console.WriteLine("9 Describe Columns");
            Console.WriteLine("10 Write your own query!");
            Console.WriteLine("11 Close connection and exit program");
            Console.WriteLine("****************************END
MENU****************************");
            int menuOption = int.MaxValue;
            try
            {
                try
                {
                    menuOption = Int32.Parse(Console.ReadLine().Trim());
                }
                catch (Exception intError)
                {
                    Console.WriteLine("That wasn't an integer value, please try again");
                    this.printMenuReadInput();
                }
            }
            catch (Exception IOerror)
            {
                Console.WriteLine("an I/O error occurred, please try again... ");
                this.printMenuReadInput();
            }
            if (!(this.executeQuery(menuOption)))
            {
                Console.WriteLine("Oops, that menu option is not available, please try again.\n");
                this.printMenuReadInput();
            }

        }

        private bool executeQuery(int menuOption)
        {
            //big switch statement
            //could have been implemented with indexing
            //on arraylist but decided against it
            //because the console app is less connected to the project
            switch (menuOption)
```

```
    {
        case 1:
            query1();
            return true;
        case 2:
            query2();
            return true;
        case 3:
            query3();
            return true;
        case 4:
            query4();
            return true;
        case 5:
            query5();
            return true;
        case 6:
            query6();
            return true;
        case 7:
            query7();
            return true;
        case 8:
            descTables();
            return true;
        case 9:
            descColumns();
            return true;
        case 10:
            writeYourOwnQuery();
            return true;
        case 11:
            endProgram();
            return true;
        default:
            return false;
    }
}

private void query1()
{
    String cmdText = "select worker.last_name, worker.worker_id, worker.worker_utility_branch_id
as utility_branch, equipment.description, equipment_out.count from equipment, worker join
equipment_out on worker.worker_id = equipment_out.equipment_out_worker_id where
equipment_out.equipment_out_equipment_id = equipment.equipment_id;";
    standardQuery(cmdText);
}
```

```
	private void query2()
	{
	    String cmdText = "select w.worker_utility_branch_id as utility_branch, w.worker_id, 
address.street, address.city, address.state, address.zip_code as zip, address.phone_number as phone 
from worker w join address on w.worker_address_id = address.address_id order by 
w.worker_utility_branch_id;";
	    standardQuery(cmdText);
	}

	private void query3()
	{
	    String cmdText = "select utility.*, task.task_id, task.appointment_date from utility left join task on 
utility.utility_id = task.task_utility_id;";
	    standardQuery(cmdText);
	}

	private void query4()
	{
	    String cmdText = "select u.description, u.longitude, u.latitude, w.worker_id, w.last_name, 
t.appointment_date from utility u, worker w, task t, task_schedule s where DATE(t.appointment_date) = 
CURRENT_DATE and " +
			"s.schedule_worker_id = w.worker_id and s.schedule_id = t.task_schedule_id and 
t.task_utility_id = u.utility_id;";
	    standardQuery(cmdText);
	}

	private void query5()
	{
	    String cmdText = "select u.description, u.utility_id, u.latitude, u.longitude, e.description as 
equip_req_desc, er.quantity as amt_needed from utility u, task t, equipment e, equipment_required er 
where u.utility_id = t.task_utility_id and er.equipment_required_task_id = t.task_id and 
er.equipment_required_equipment_id = e.equipment_id;";
	    standardQuery(cmdText);
	}

	private void query6()
	{
	    String cmdText = "select w.first_name, w.last_name, u.description, t.task_id, t.appointment_date 
from worker w, utility u, task t, task_schedule s where s.schedule_worker_id = w.worker_id and 
t.task_schedule_id = s.schedule_id and u.utility_id = t.task_utility_id;";
	    standardQuery(cmdText);
	}

	private void query7()
	{
	    String cmdText = "select u.utility_branch_name, u.utility_branch_id, e.description, e.quantity, 
eo.count as quantityOut from equipment e, equipment_out eo, utility_branch u where
```

```
u.utility_branch_id = e.equipment_utility_branch_id and eo.equipment_out_equipment_id =
e.equipment_id;";
            standardQuery(cmdText);
        }

        private void descTables()
        {
            MySqlConnection con = new MySqlConnection(connectionString);
            try
            {
                con.Open();
                MySqlCommand command = con.CreateCommand();
                command.CommandText = "SHOW TABLES;";
                MySqlDataReader Reader;
                Reader = command.ExecuteReader();
                while (Reader.Read())
                {
                    string row = "";
                    for (int i = 0; i < Reader.FieldCount; i++)
                        row += Reader.GetValue(i).ToString() + ", ";
                    Console.WriteLine(row);
                }
                printMenuReadInput();
            }
            catch (MySqlException msqle)
            {
                Console.WriteLine("There was an error with the connection, please try again...");
                printMenuReadInput();
            }
            finally
            {
                if (con != null)
                {
                    con.Close();
                }
            }
        }

        private void descColumns()
        {
            Console.WriteLine("Type the table name or exit to exit the desc columns option...");
            String tblName = Console.ReadLine();
            if (tblName == "exit")
            {
                printMenuReadInput();
            }
            MySqlConnection con = new MySqlConnection(connectionString);
            try
            {
                con.Open();
                MySqlCommand command = con.CreateCommand();
                command.CommandText = "SHOW FIELDS FROM " + tblName;
                MySqlDataReader Reader;
                Reader = command.ExecuteReader();
                while (Reader.Read())
                {
                    string row = "";
                    for (int i = 0; i < Reader.FieldCount; i++)
                    row += Reader.GetValue(i).ToString() + ", ";
                    Console.WriteLine(row);
                }
                printMenuReadInput();
            }
            catch (MySqlException msqle)
            {
                Console.WriteLine("There was an error with the connection or the table does not exist, please
try again...");
                descColumns();
            }
            finally
            {
                if (con != null)
                {
                    con.Close();
                }
            }
            printMenuReadInput();
        }

        private void standardQuery(string query)
        {
            MySqlConnection con = new MySqlConnection(connectionString);
            MySqlDataReader reader = null;
            try
            {
                con.Open();
                Console.WriteLine();
                DateTime start;
                TimeSpan time;
                start = DateTime.Now;
                MySqlCommand cmd = new MySqlCommand(query, con);
                reader = cmd.ExecuteReader();
                time = DateTime.Now - start;
                for (int i = 0; i < reader.FieldCount; i++)
                {
                    Console.Write("{0, -20}", reader.GetName(i));
                }
                Console.WriteLine();
                while (reader.Read())
                {
                    for (int j = 0; j < reader.FieldCount; j++)
                    {
                        try {
                            Console.Write("{0, -20}", reader.GetString(j));
                        }
                        catch (Exception msqle)
                        {
                            Console.Write("{0, -20}", "null");
                        }
                    }
                    Console.WriteLine();
                }
                Console.WriteLine("Query Completed in..." + time.TotalMilliseconds + "ms");
                printMenuReadInput();
            }
            catch (MySqlException msqle)
            {
                Console.WriteLine(msqle.ToString());
                Console.WriteLine("There was an error with the connection, please try again...");
                printMenuReadInput();
            }
            finally
            {
                if (con != null)
                {
                    con.Close();
                }
            }
        }

        /**we make an assumption here that a user would check the counts of current values in equipment
for a utility branch before adding a row to equipment out or
        updating it. this project was more about the mysql then it was about the console app so we left this
part out and if we had another day or two for the project
        we would've added this**/
        private void customQuery(string query)
        {
            MySqlConnection con = new MySqlConnection(connectionString);
            MySqlDataReader reader = null;
                try
                {
                con.Open();
                Console.WriteLine();
                DateTime start;
                TimeSpan time;
                start = DateTime.Now;
                MySqlCommand cmd = new MySqlCommand(query, con);
                reader = cmd.ExecuteReader();
                time = DateTime.Now - start;
                for (int i = 0; i < reader.FieldCount; i++)
                {
                    Console.Write("{0, -20}", reader.GetName(i));
                }
                Console.WriteLine();
                while (reader.Read())
                    {
                    for (int j = 0; j < reader.FieldCount; j++)
                    {
                        try
                        {
                            Console.Write("{0, -20}", reader.GetString(j));
                        }
                        catch (Exception msqle)
                        {
                            Console.Write("{0, -20}", "null");
                        }
                    }
                    Console.WriteLine();
                    }
                Console.WriteLine("Query Completed in..." + time.TotalMilliseconds + "ms");
                printMenuReadInput();
                }
                catch (MySqlException msqle)
                {
                Console.WriteLine(msqle.ToString());
                Console.WriteLine("There was an error with your sql syntax or the connection, please try
again...");
                writeYourOwnQuery();
                }
                finally
                {
                    if (con != null)
                    {
                        con.Close();
                    }
                }

        }

        private void writeYourOwnQuery()
        {
            Console.WriteLine("Please type MySQL syntax to execute a query of your choosing...type exit; to
go back to the menu");
            String query = "**";
            ConsoleKeyInfo keyinfo;
            while (true)
            {
                keyinfo = Console.ReadKey();
                if (query.Substring(query.Length - 1) == ";" && (keyinfo.Key.Equals(ConsoleKey.Enter)))
                {
                    break;
                }
                else if (keyinfo.Key.Equals(ConsoleKey.Enter))
                {
                    Console.WriteLine();
                    Console.Write("\t>:");
                }
                else
                {
                    query += keyinfo.KeyChar;
                }
            }
            query = query.Remove(0, 2);
            if (query.Equals("exit;"))
            {
                Console.WriteLine("Exiting custom query menu option, printing menu...");
                printMenuReadInput();
            }
            customQuery(query);
        }

    }
    }
```

1. Team Name: Street Sharks
2. Tuesdays/Thursdays 5:30-7:30pm
3. Contract complete.

Project Proposal

Iteration II

- What is the application area of the database?
    - Tool for tracking city utility maintenance schedules (related to the electrical grid)
- What kind of data will the database hold? (in general, don't need to list every attribute yet)
    - UTILITY_BRANCH
        - ub_id
        - name
        - ub_address_id
    - ADDRESS
        - address_id
        - street
        - city
        - state
        - zip_code
        - phone_number
    - WORKER
        - worker_id
        - worker_ub_id
        - worker_address_id
        - fname
        - lname
        - wage
    - SCHEDULE
        - schedule_id
        - schedule_worker_id
        - start_date
        - end_date
    - TASK
        - task_id
        - task_utility_id
        - task_schedule_id
        - appointment_date
    - UTILITY
        - utility_id
        - description

        - longitude
        - latitude
    - EQUIPMENT
        - equipment_id
        - equipment_ub_id
        - count
        - description
    - EQUIPMENT_REQUIRED
        - equipment_required_task_id
        - equipment_required_equipment_id
        - count
    - EQUIPMENT_OUT
        - equipment_out_ub_id
        - equipment_out_equipment_id
        - equipment_out_worker_id
        - count

- What kinds of queries will you be able to answer?
    - Don't need to write SQL statements at this point but to give descriptions in English about sample queries in the database, including interesting queries involving multiple tables, aggregation, and/or grouping, and describe how they support the application area that you choose.
        - Show me the equipment needed to complete Task X at Light post Y
        - Show me the schedule for Employee X
        - Show me the number of employees at Branch X who have certain tasks
        - Show me the schedule for completing Task X
        - Show me all branches and their address and employees
        - Show me all the tasks that need to be completed by a certain date X
            - Supports the Application Area because: This database application contains the data for a Utility Branch to perform its function.
- Who will work on each part of the project? Can be "all."
    - ALL

Test data methodology

Wanted multiple utility branches with a variety of related records from each other entity. For example, some utility branches will have no employees, 1 employee, or many employees. By extension, some utilities will be associated with 0, 1, or many tasks. This logic applies to all entities and their related entities.

Why we commented triggers

Apparently, Azure uses ClearDB and unless the server is paid for, SUPER privileges cannot be assigned, meaning we cannot execute triggers on the server without a paid subscription.

https://getsatisfaction.com/cleardb/topics/create_function_without_super_privileges

See top response.
Early scripts used to create console application with DB
Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using streetsharksconsoleapp;

namespace streetsharksconsoleapp
{
    class Program
    {
        static void Main(string[] args)
        {
            Execute();
        }

        private static void Execute()
        {
            StreetSharksConnection db = new StreetSharksConnection();
            db.openConnection();
            db.printMenuReadInput();
        }
    }
}
```

StreetSharksConnection.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using MySql.Data.MySqlClient;

namespace streetsharksconsoleapp
{
  class StreetSharksConnection
  {
      private static StreetSharksConnection singletonInstance; //stops the program from having multiple
instantiatons of the same connection
      private String connectionString = "Database=streetsharksutilitiesdb;Data Source=us-cdbr-azure-
west-c.cloudapp.net;User Id=bbda996f99fe96;Password=02b47f27";
      private MySqlConnection con;

      public StreetSharksConnection() {}

      public static StreetSharksConnection Instance //does not work for multi threading situations
      {
        get
        {
          if (Instance == null)
          {
            singletonInstance = new StreetSharksConnection();
          }
          return singletonInstance;
        }
      }

        public void openConnection()
        {
          try
          {
            con = new MySqlConnection(connectionString);
            con.Open(); //open the connection
            Console.WriteLine("Street sharks utilitiesdb opened...");

        }
          catch (MySqlException error)
          {
            Console.WriteLine("Error connecting to the database, printing error: " + error.ToString());
            Console.WriteLine("Printing stack trace: " + error.StackTrace);
          }
        }
        private void endProgram()
        {
            Console.WriteLine("Closing connection...thanks for using the streetsharks utilitiesdb!");
            if (con != null)
            {
                con.Close();
            }
        }

        public void printMenuReadInput()
        {
            Console.WriteLine("StreetSharks Database Menu\nEnter a menu number option to continue");

Console.WriteLine("*****************************MENU******************************");
            Console.WriteLine("1 Query 1\t2 Query 2\t3 Query 3\n4 Query 4\t5 Query 5\t6 Query 6\n7
Query 7\t8 Desc Tbls\t9 Desc Cols\n10 Write your own query!\n11 Close connection and exit program");

Console.WriteLine("*****************************MENU******************************");
            int menuOption = int.MaxValue;
            try
            {
                try
                {
                    menuOption = Int32.Parse(Console.ReadLine().Trim());
                }
                catch (Exception intError)
                {
                    Console.WriteLine("That wasn't an integer value, please try again");
                    this.printMenuReadInput();
                }
            }
            catch (Exception IOerror)
            {
                Console.WriteLine("an I/O error occurred, please try again... ");
                this.printMenuReadInput();
            }
            if (!(this.executeQuery(menuOption)))
            {
                Console.WriteLine("Oops, that menu option is not available, please try again.\n");
                this.printMenuReadInput();
            }

        }

      private bool executeQuery(int menuOption)
      {
        switch (menuOption)
        {
```

```
        case 1:
            query1();
            return true;
        case 2:
            query2();
            return true;
        case 3:
            query3();
            return true;
        case 4:
            query4();
            return true;
        case 5:
            query5();
            return true;
        case 6:
            query6();
            return true;
        case 7:
            query7();
            return true;
        case 8:
            descTables();
            return true;
        case 9:
            descColumns();
            return true;
        case 10:
            writeYourOwnQuery();
            return true;
        case 11:
            endProgram();
            return true;
        default:
            return false;
    }
}

private void query1()
{
    Console.WriteLine("Entering query 1...");
    this.printMenuReadInput();
}

private void query2()
{
    Console.WriteLine("Entering query 2...");
    this.printMenuReadInput();
```

```
}

private void query3()
{
   Console.WriteLine("Entering query 3...");
   this.printMenuReadInput();
}

private void query4()
{
   Console.WriteLine("Entering query 4...");
   this.printMenuReadInput();
}

private void query5()
{
   Console.WriteLine("Entering query 5...");
   this.printMenuReadInput();
}

private void query6()
{
   Console.WriteLine("Entering query 6...");
   this.printMenuReadInput();
}

private void query7()
{
   Console.WriteLine("Entering query 7...");
   this.printMenuReadInput();
}

private void descTables()//almost done
{
   try
   {
      MySqlCommand command = con.CreateCommand();
      command.CommandText = "SHOW TABLES;";
      MySqlDataReader Reader;
      Reader = command.ExecuteReader();
      while (Reader.Read())
      {
         string row = "";
         for (int i = 0; i < Reader.FieldCount; i++)
            row += Reader.GetValue(i).ToString() + ", ";
         Console.WriteLine(row);
      }
      printMenuReadInput();
```

```
            }
            catch (MySqlException msqle)
            {
                Console.WriteLine("There was an error with the connection, please try again...");
                printMenuReadInput();
            }
        }

        private void descColumns()
        {
            Console.WriteLine("Type the table name...");
            String tblName = Console.ReadLine();
            Console.WriteLine("Entering desc columns...");
            this.printMenuReadInput();
        }

        private void customQuery(string query)
        {


            //Do something here

            // try
            //{
            /*MySqlCommand cmd = new MySqlCommand(query, con);
            cmd.ExecuteNonQuery();
            Console.WriteLine("------------------------------End Output------------------------------");
            this.printMenuReadInput();*/
            MySqlDataReader reader = null;
                try
                {
                Console.WriteLine();
                DateTime start;
                TimeSpan time;
                start = DateTime.Now;
                MySqlCommand cmd = new MySqlCommand(query, con);
                    reader = cmd.ExecuteReader(); //execure the reader
                                    /*The Read() method points to the next record It return false if there are no
more records else returns true.*/
                    while (reader.Read())
                    {
                    /*reader.GetString(0) will get the value of the first column of the table myTable because we
selected all columns using SELECT * (all); the first loop of the while loop is the first row; the next loop
will be the second row and so on...*/
                    Console.WriteLine(reader.GetString(0));
                    }
                time = DateTime.Now - start;
                Console.WriteLine("Query Completed in..." + time.TotalMilliseconds + "ms");
            printMenuReadInput();
            }
         catch (MySqlException msqle)
         {
            //Console.WriteLine(msqle.ToString());
            Console.WriteLine("There was an error with your sql syntax or the connection, please try
again...");
            writeYourOwnQuery();
         }

      }

      private void writeYourOwnQuery()
      {
         Console.WriteLine("Please type MySQL syntax to execute a query of your choosing...type exit; to
go back to the menu");
         String query = "**";
         ConsoleKeyInfo keyinfo;
         while (true)
         {
            keyinfo = Console.ReadKey();
            if (query.Substring(query.Length - 1) == ";" && (keyinfo.Key.Equals(ConsoleKey.Enter)))
            {
               break;
            }
            else if (keyinfo.Key.Equals(ConsoleKey.Enter))
            {
               Console.WriteLine();
               Console.Write("\t>:");
            }
            else
            {
               query += keyinfo.KeyChar;
            }
         }
         query = query.Remove(0, 2);
         if (query.Equals("exit;"))
         {
            Console.WriteLine("Exiting custom query menu option, printing menu...");
            printMenuReadInput();
         }
         customQuery(query);
      }

   }
   }
```

1. Team Name: Street Sharks
2. Tuesdays/Thursdays 5:30-7:30pm
3. Contract complete.

Project Proposal

Iteration I

- What is the application area of the database?
    - Tool for tracking city utility maintenance schedules (related to the electrical grid)
- What kind of data will the database hold? (in general, don’t need to list every attribute yet)
    - Utility Branch
        - Branch Num
        - Branch Name
        - Other Foreign Keys
    - Location
        - City
        - State
        - Street
        - Country
    - Worker
        - Staff Num
        - Other Foreign Keys
    - Schedule
        - Other Foreign Keys
    - Task
        - Time
        - Other Foreign Keys
    - Light Post
        - Light Post Num
        - Status
        - Latitudinal/Longitudinal Coordinates
        - Other Foreign Keys
    - Transformer Box
        - Transformer Box Num
        - Status
        - Latitudinal/Longitudinal Coordinates
        - Other Foreign Keys
    - Vehicle
        - Vehicle Num
        - Make

        - Model
        - Year
        - Other Foreign Keys
    - Equipment
        - Equipment Num
        - Other foreign keys
- What kinds of queries will you be able to answer?
    - Don't need to write SQL statements at this point but to give descriptions in English about sample queries in the database, including interesting queries involving multiple tables, aggregation, and/or grouping, and describe how they support the application area that you choose.
        - Show me the equipment needed to complete Task X at Light post Y
        - Show me the schedule for Employee X
        - Show me the number of employees at Branch X who have vehicles
        - Show me the schedule for completing Task X
        - Show me all branches and their locations
        - Show me all the tasks that need to be completed by a certain date X
            - Supports the Application Area because: This database application contains the data for a Utility Branch to perform its function.
- What is your schedule for completing the deliverables listed in the "Specific deliverables" Section?
    - We plan to meet every Tuesdays/Thursdays from 5:30pm-7:30PM as needed to complete an iteration by the scheduled due date. If due dates are upcoming and we suspect we will be unable to complete an iteration, we will spend Saturdays until we complete the project iteration.
- Who will work on each part of the project? Can be "all."
    - ALL