SCT Electrical Team Database

Implementation Manual

Ruicheng Xu, William Lorey, M. Jacob Schoonover

Table of Contents

Problem statement	**4**
System requirements	**4**
Conceptual database design	**5**
Functional requirements	**6**
Logical database design	**7**
Application Programs	**13**
User interface design	**17**
Implementation plan	**24**
Code listing	**24**
Sample output	**25**

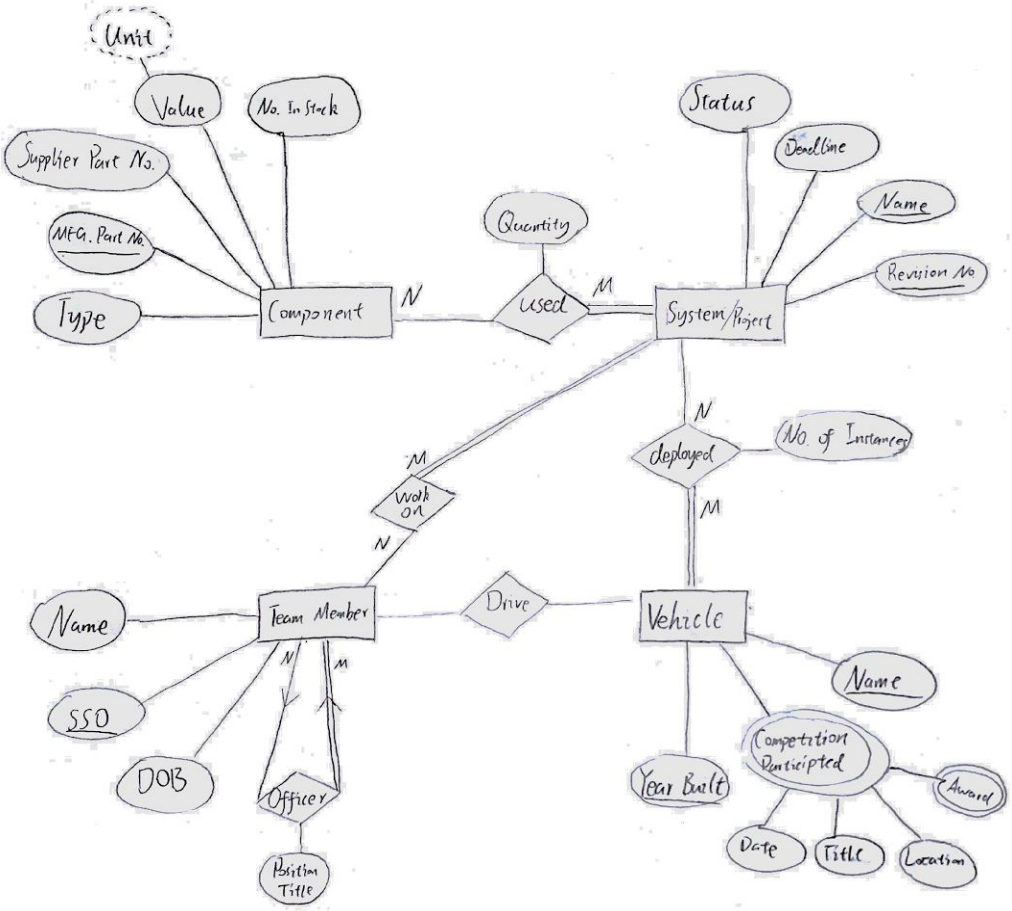
Problem statement

While working in a collegiate design team environment, the organization of the team needs to be held in paramount. This includes components, projects, team members, and vehicles (in the case of the S&T Solar Car Electrical Subteam (SCEST)). Often times taking care of such matters is not held to a high enough standard due to time and manpower constraints. The proposed application will contain the aggregation of data pertaining to the aforementioned aspects of the SCEST.

System requirements

The proposed system will act as a catalog for all electrical components used by the SCEST. Included for each component will be a type, manufacturer part number, supplier part number, a value (in Ohms, Henries, Farads, etc.), and number in stock. In addition, the projects currently being worked on by the SCEST are to be stored along with information about their status, deadline, name, revision number, and how many instances of that project appear on the vehicle. Data pertaining to team members such as name, SSO, DOB, and what projects they work on is to be stored. Lastly, vehicle information is stored. This includes the year completed, competition(s) participated in, projects incorporated on the car, drivers of the car, name, and awards received. The system will be used on a thrice-weekly basis to track the status and progress of the SCEST in their endeavors.

Conceptual database design



Functional requirements

Retrievals

- Get every component with a given value. Returns a component tuple. Interacts with the component table.
- Get every team member working on a given project. Returns a team member tuple. Interacts with the team member table and the system/project table.
- Get the status of a system/project. Returns the status of the given project. Interacts with the system/project table.
- Get the position of a team member. Returns the position of a given team member. Interacts with the team member table.
- Get all drivers of a given vehicle. Returns the names of all drivers of a given vehicle. Interacts with the vehicles table and the team member table.
- Get every vehicle that has raced at a specific location. Returns names of vehicles. Interacts with the vehicle table.
- Get the team members who have birthdays in a given month. Returns team member names. Interacts with the team member table.

Updates

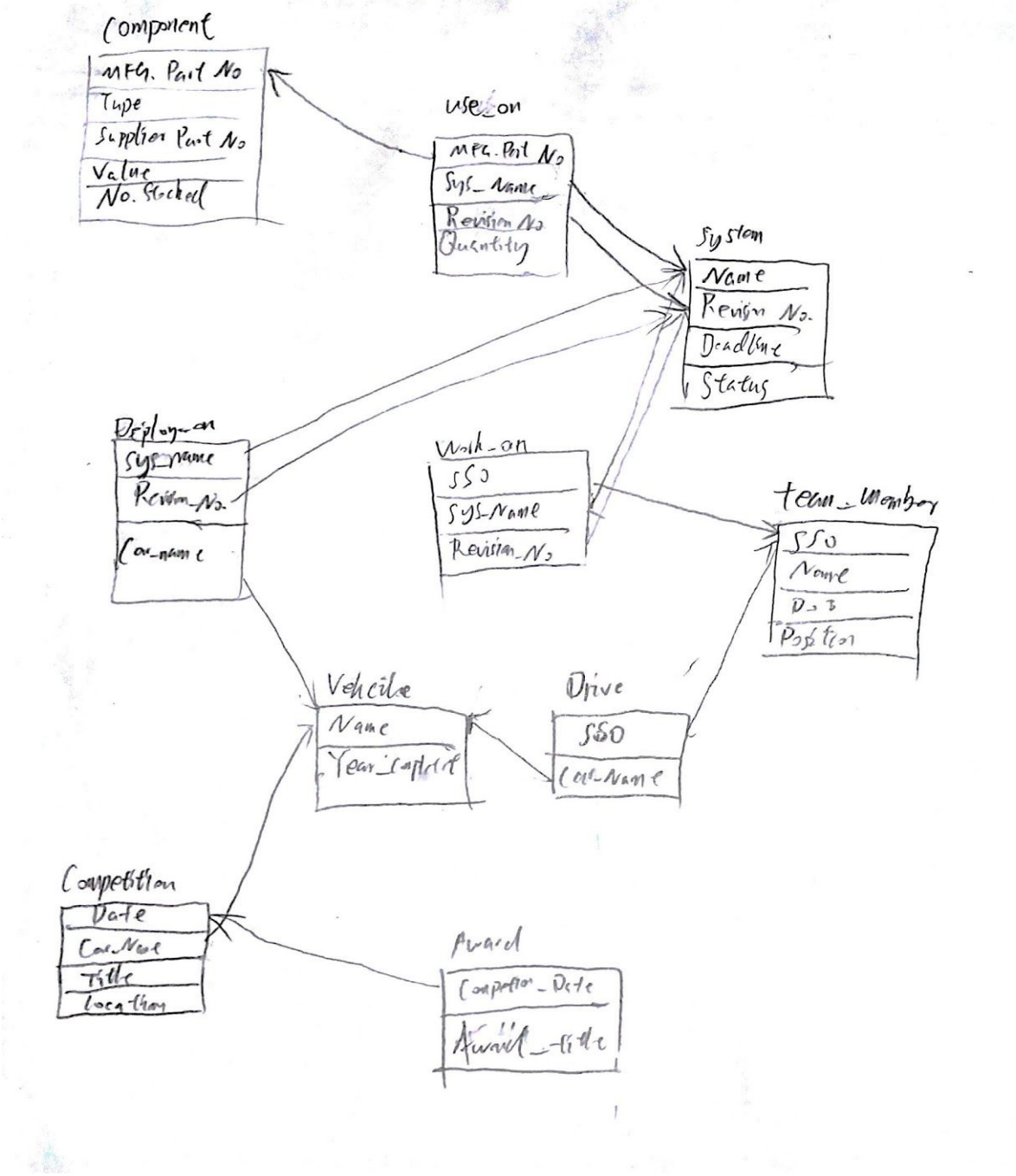
- Delete the drivers of a vehicle. Interacts with the team member table and the vehicle table.
- Add a member of the team, populating their information in the team member table, making sure to include the SSO of the team member to uphold the database requirements.
- Delete a team member. Interacts with the vehicle table to remove the team member as a driver if that is the case then removes the team member from the team member table as well as the project table.
- Add a new component to the component table, populating its information and adding it to a system/project by interacting with the system/project table. The manufacturer part number of the component must be included to uphold the database requirements.
- Add a system to the system table, interacting with the component table and the vehicle table to make the update and ensure the data is consistent.
- Add a new race and award. This update interacts with the awards table as well as the vehicle table.

The report requirements are listed in the above interactions. They are dependent on the type of information requested. Data entry requirements are also included above and depend on the specific interaction with the application.

Note: some updates listed above must be completed by administrators in order to keep user interactivity in check.

Logical database design

Foreign key references: Arrows indicate foreign key references. They point from the primary key (PK) of a table to the foreign key (FK) that particular attribute represents in another table.



SQL Table and Attribute description

The underlined attribute(s) make up the primary key of their respective table. Foreign keys are defined in the above diagram.

Table Name	**Attribute (Primary keys are underlined)**	**Datatype**	**Description**
VEHICLE	NAME	Varchar(20)	The name of the solar car. This uniquely identifies each car entry. PK of VEHICLE.
	YEAR_COMPLETED	int(4)	The year the solar car is finished in.
COMPONENT	MFG_PART_NO	varchar(25)	The manufacturer part number of the component. This collection of numbers and characters is unique for all components across vendors. PK of COMPONENT.
	TYPE	varchar(3)	The component type refers to RES (resistor), CAP (capacitor), IND (inductor), or OTH (other). NOT NULL.
	SUPP_PART_NO	varchar(25)	The supplier part number helps in ordering parts from different companies.
	NO_STOCK	int(5)	Number of components in stock of that particular component.
DEPLOY_ON	SYS_NAME	varchar(30)	Name of the system on the car. FK of SYSTEM combined with REV_NO.

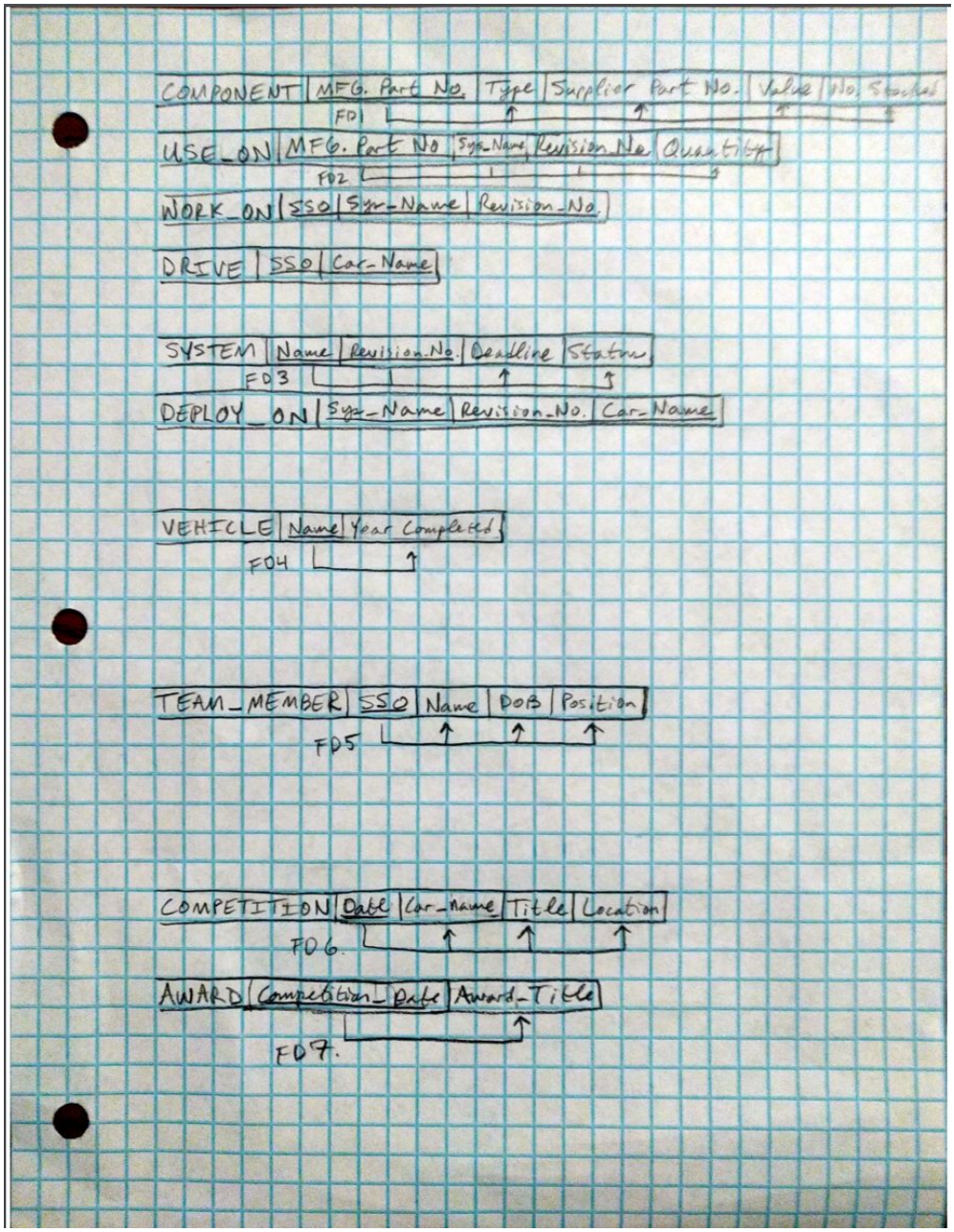
	REV_NO	varchar(2)	Revision number of the system. FK of SYSTEM combined with SYS_NAME.
	CAR_NAME	varchar(42)	Name of the car which a system is implemented on. This along with the SYSTEM primary key (SYS_NAME and REV_NO) uniquely identifies a tuple in the DEPLOY_ON table. FK of VEHICLE.
DRIVE	SSO	varchar(7)	The uniquely identifying student number.
	CAR_NAME	varchar(42)	The name of the solar car, which is driven by a student. This along with the SSO uniquely identifies a tuple in the DRIVE table.
SYSTEM	NAME	varchar(30)	The name of a particular system on the car.
	REV_NO	varchar(2)	The revision number of the system. This along with the NAME uniquely identifies a system.
	DEADLINE	date	The deadline for the system.
	STATUS	varchar(11)	The status of the project (completed, bad, or in progress). NOT NULL.
TEAM_MEMBER	SSO	varchar(7)	The single sign on of a team member. This

			uniquely identifies a team member.
	NAME	varchar(30)	The name of a team member (first and last name). NOT NULL.
	DOB	date	The date of birth of a team member.
	POSITION	varchar(30)	The position a team member holds on the team (Team member, Electrical Lead, etc.).
USE_ON	MFG_PART_NO	varchar(25)	The manufacturer part number of a component. FK, references to COMPONENT MFG_PART_NO.
	SYS_NAME	varchar(30)	The name of the system. FK to SYSTEM along with REV_NO.
	REV_NO	varchar(2)	The revision number of a system. This FK references the SYSTEM table along with the SYS_NAME. NOTE: The REV_NO, SYS_NAME, and MFG_PART_NO make up the PK of USE_ON.
	QUANTITY	int(5)	The quantity of parts used on a system.
WORK_ON	SSO	varchar(7)	The single sign on of a team member. This uniquely identifies a team member. FK of the TEAM_MEMBER table.

	SYS_NAME	varchar(30)	The name of the system. FK to SYSTEM along with REV_NO.
	REV_NO	varchar(2)	The revision number of a system. This FK references the SYSTEM table along with the SYS_NAME. NOTE: SSO, SYS_NAME, and REV_NO make up the primary key of the WORK_ON table.
COMPETITION	DATE	date	The date on which the competition starts. This uniquely identifies the competition, as competitions don't ever happen at the same time.
	CAR_NAME	varchar(42)	The name of the car competing in the competition.
	TITLE	varchar(30)	The title of the competition.
	LOCATION	varchar(30)	The location of the competition.
AWARD	COMPETITION_DATE	date	The date on which a competition starts. This is a FK of the COMPETITION table and PK of the AWARD table along with AWARD_TITLE.
	AWARD_TITLE	varchar(30)	The award that is obtained from a competition. Multiple awards can be won from a single

			competition, therefore the awards are given their own table. PK of the AWARD table along with the COMPETITION_DATE.

Functional dependencies diagram: The following diagram represents the functional dependencies of each relation as depicted by arrows. The tables are proven to be in 3NF (See the statements below the pictorial diagram).



1NF:all the attributes in a relation must have atomic domains
2NF: if X → A holds, then there should not be any proper subset Y of X, for which Y → A also holds true.
3NF(Third Normal Form):
2NF and

- No non-prime attribute is transitively dependent on prime key attribute.
- For any non-trivial functional dependency, X → A, then either −
    - X is a superkey or, A is prime attribute.

All relations are in 3NF. The following demonstrates the relation 3NF checks:

VEHICLE	Name is a superkey.
COMPONENT	MFG_PART_NO is a superkey.
DEPLOY_ON	SYS_NAME and REV_NO together is a superkey, no partial dependency(2NF).
DRIVE	SSO and CAR_NAME together is a super key, and fits 2NF and 1NF.
SYSTEM	NAME and REV_NO together is a super key, and fits 2NF and 1NF.
TEAM_MEMBER	SSO is a super key. and fits 2NF and 1NF.
USE_ON	3NF checked.
WORK_ON	3NF checked.
COMPETITION	3NF checked DATE is a superkey.
AWARD	COMPETITION_DATE and AWARD_TITLE together is a superkey, 3NF checked

BCNF: All relations are in BCNF (dependencies each have their own table).

Application Programs

The application is built using a mySQL database, PHP backend, and displayed to the user and made interactive using web languages (HTML, CSS, and JavaScript).
Modules and their transactions contained in the application are as follows:

Get user input
This module interacts with the application front-end and gets information from the user about any particular facet of the current query/database update being performed.
Input parameters are the current table the user is viewing/altering (obtained based on the current page the user is on), and any pertinent data about the update/retrieval from the user (i.e. attribute names and their values).
The output of course depends on the nature of the user input. This module itself does not explicitly return any values to the user, but the modules with which it interacts do (see other modules for more in-depth information).
This module interacts with the prepare SQL statement module, add a record module, and remove a record module.
Side effects include the entry of user input into the database system.
Pseudocode overview: Determine what type of user input is required for a certain transaction. Display the necessary text fields, drop down menus, etc. to receive the data from users. Prepare a SQL statement for the data and submit it to the database (via a foreign module).

Add a record
This module will be present in many forms: adding cars to the database, adding team members, adding systems, etc.
Input parameters are the table a user wishes to insert into and the tuple they wish to insert (obtained from the get user input module). These values are presented in a manner by the interface such that they appear as forms tying to different entities of the database.
There are no output parameters included other than a "success" message displayed to the user once a record is entered into the database, or "fail" if that is the case.
This module connects to the get user input module, and the prepare SQL statement module.
Side effects of this module's execution include the successful addition of a tuple to the user's desired table. Of course, their input must be validated at the time of entry.
Pseudocode overview: Connect to the database (if not already connected). Get input from user. Validate user input and get more input if needed. Create SQL statement for insertion of the tuple. Insert into the SQL database.

Remove a record
This module removes a record (tuple) from a given database. Note: this feature will not be available to the main application users, but will be used for easy application upkeep and maintenance.
Input parameters include the primary key of the tuple to be deleted along with the table in which that tuple is stored.
Out parameters include a "success" message, or "fail" message, depending on the outcome of the alteration.
This module interacts with the get user input module and prepare SQL statement module.
Side effects of this module's running include the removal of a tuple from a table in the database.Dis
Pseudocode overview: Ask the user which record to delete from which table in the database. Connect to the database and prepare a SQL statement for the required update before sending the update to the database. Field the success/fail of the update from the database.

Prepare SQL statement
This module creates SQL statements for use with database interaction. It does most of the heavy lifting regarding database transactions.
Input parameters include the type (query, update, etc.), amount of parameters (attributes) relating to the statement, the attributes themselves (retrieved from the user), and any conditions for the statement.
The output is the return of the query (if applicable) or nothing, depending on the statement.
This module interacts with the add a record module, delete a record module, connect to database module, and the display data module.
Side effects of this module's execution include an alteration. extraction, or query from the database which the user can then see.
Pseudocode overview: Get input from the user pertaining to the certain query/update. Based on that user input, create the SQL statement in PHP. Open the connection to the database and send the statement over, receiving the output of whatever the statement returns and displaying it to the user through the display data module (if necessary). SQL queries needed will be in the following basic format: "SELECT [attr. list] FROM [table list] WHERE [condition list]". In addition updates will have the following format: "ALTER [table name] [ADD/ALTER COLUMN/DROP COLUMN/etc.] [attr. name and other parameters]". Any SQL statements needed in the application are written this way, or in slight variations (chosen as cookie-cutter statements and refined for the particular instance).

Connect to the database
Our PHP needs to connect to the database, hosted on the web server before anything regarding the database can be done.

Input parameters are the server name, username, password, and database name.
A PHP connection variable is returned from this module and can be used to test the connection.
This module interacts with the prepare SQL statement module.
Side effects of this module's execution include connecting our PHP file to the database.
Pseudocode overview: Create a new connection with the server name, username, password, and database name. Test the connection for errors before proceeding.
PHP code for creating a database connection and verifying it:

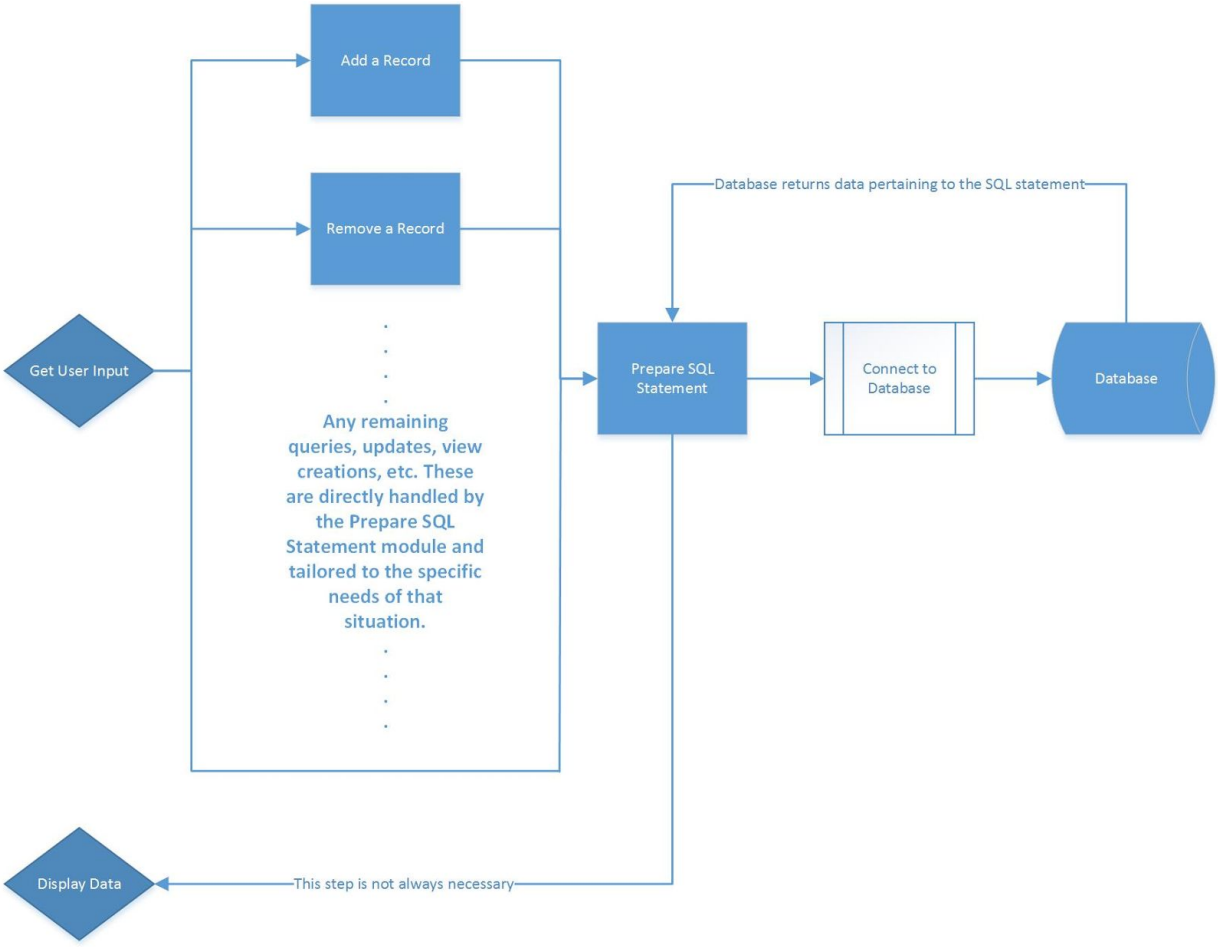
```
<?php
        // Create connection
        $conn = new mysqli($servername, $username, $password, $dbName);

        // Check connection
        if ($conn->connect_error) {
                die("Connection failed: " . $conn->connect_error);
        }
?>
```

Note: The variables $servername, $username, $password, and $dbName are declared elsewhere in the PHP code and are dependant on the database information.

Display data
This module displays data to the user about an update, query, or any database interaction that requires data to be displayed to the user.
The input is the data itself to be displayed, including the table title and attribute data to be displayed.
The output is the data displayed to the user.
This module interacts with the prepare SQL statement module.
Side effects of this module running include the user viewing information and data pertaining to the database.
Pseudocode overview: Once the prepare SQL statement module returns the data to be displayed, format it using PHP (based on the type of data, etc.) and display it to the user using HTML and CSS.

Note: All queries, updates, view creations, etc. not otherwise listed are handled by the Prepare SQL statement module for general cases and altered on a case by case basis depending on the
application needs.

Control chart (module calling sequence)



User interface design

To demonstrate the user interface, a *preliminary* test design has been made. This design consists of a web interface which links to the mySQL database using PHP on the back-end. For simplicity, screenshots of user interactions with the VEHICLE table only are shown. This is because interactions with other tables of the database will be very similar. For demonstrative purposes, the screenshots will be a step by step look at a user opening the application and entering in a new Solar Car into the database. Note: the only attributes of the VEHICLE table, NAME and YEAR_COMPLETED are interacted with in this example.

Image 1: Loading the welcome page. From the welcome page, the user can navigate to the shown tabs to perform various interactions with the database.

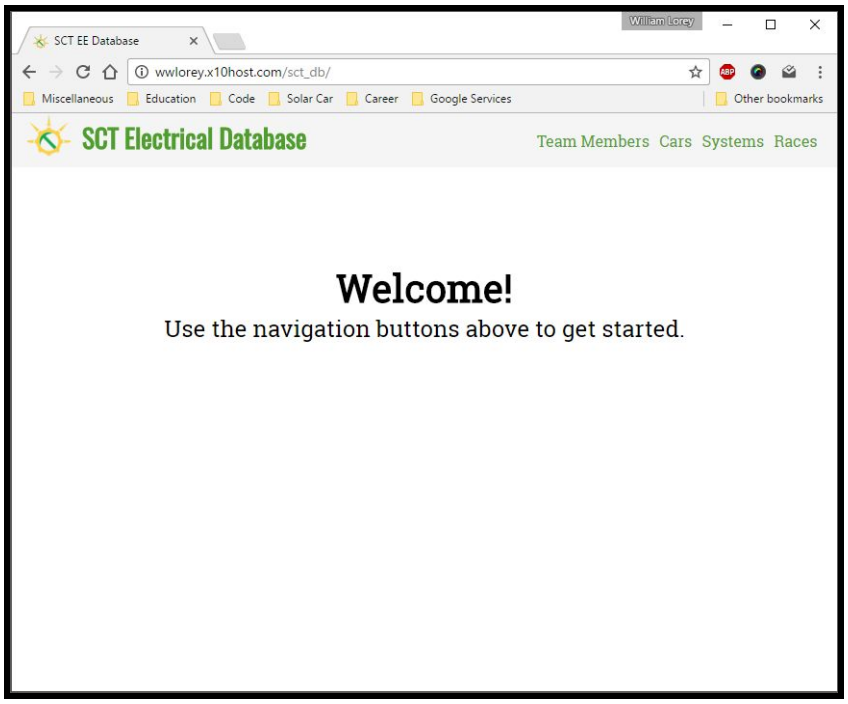


Image 2: Opening the "Cars" tab from the navigation buttons in the upper right corner of the window.

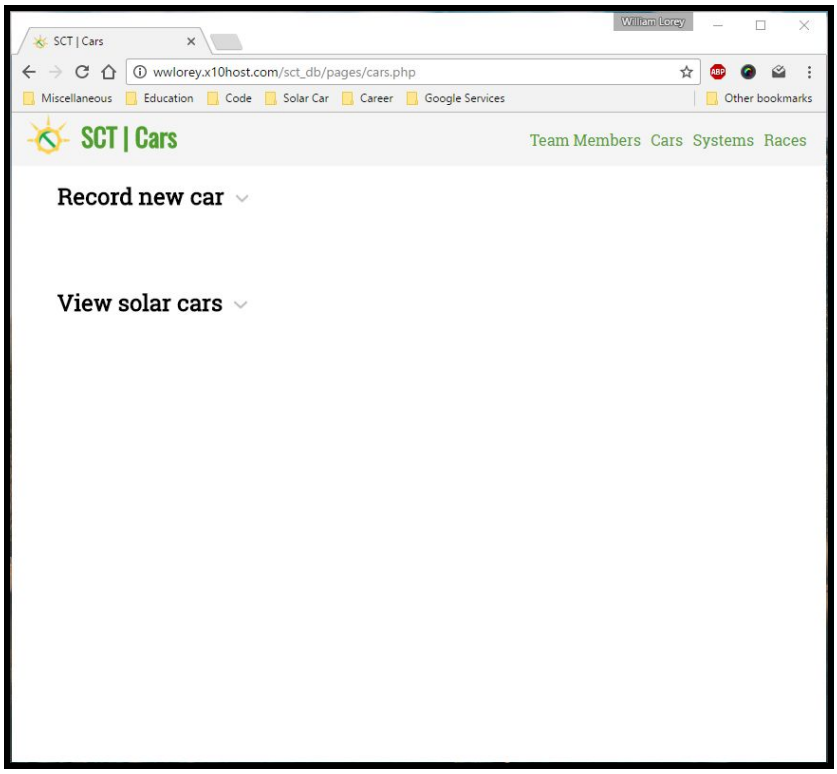


Image 3: Opening the "Record new car" dropdown form.

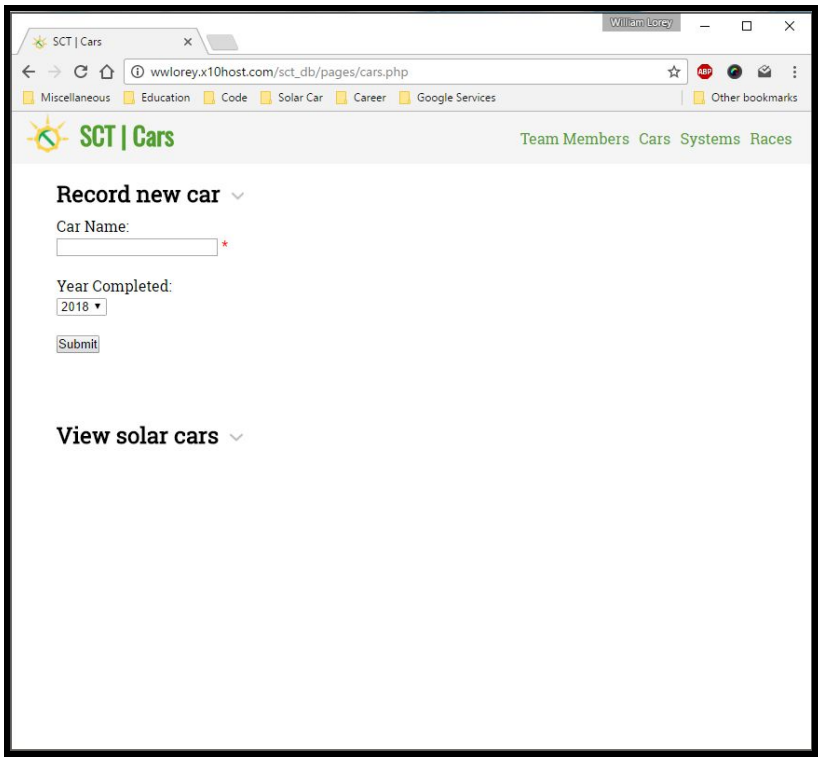


Image 4: Demonstration of table constraints: When the "Submit" button is pressed without a "Car Name", the submission is blocked. The application does not allow a user to submit a car without a "Car Name" (the NAME attribute in the VEHICLE table).

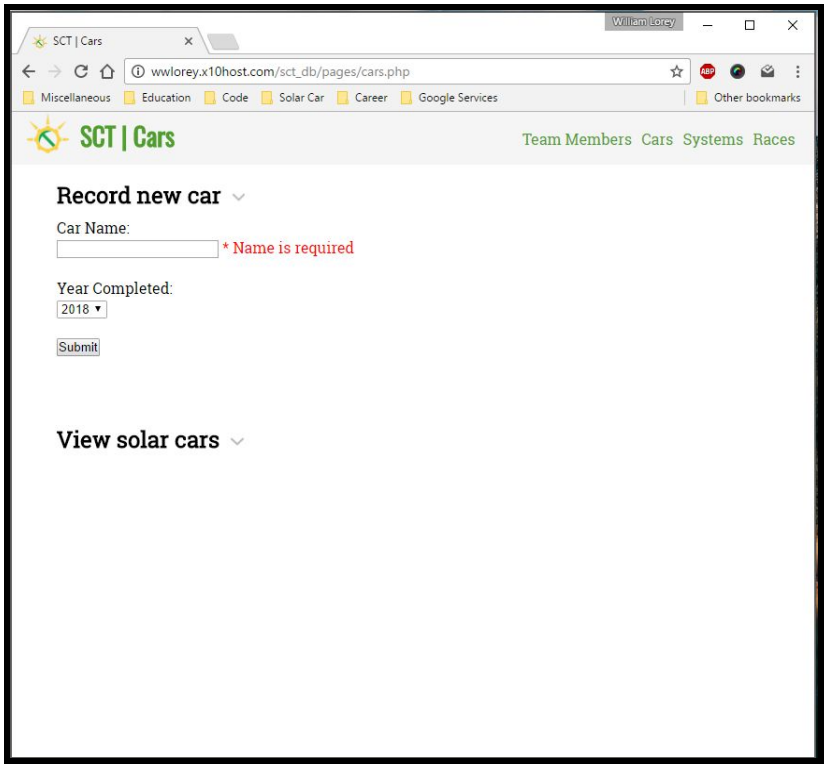


Image 5: Recording a new car and supplying a car name to the form.

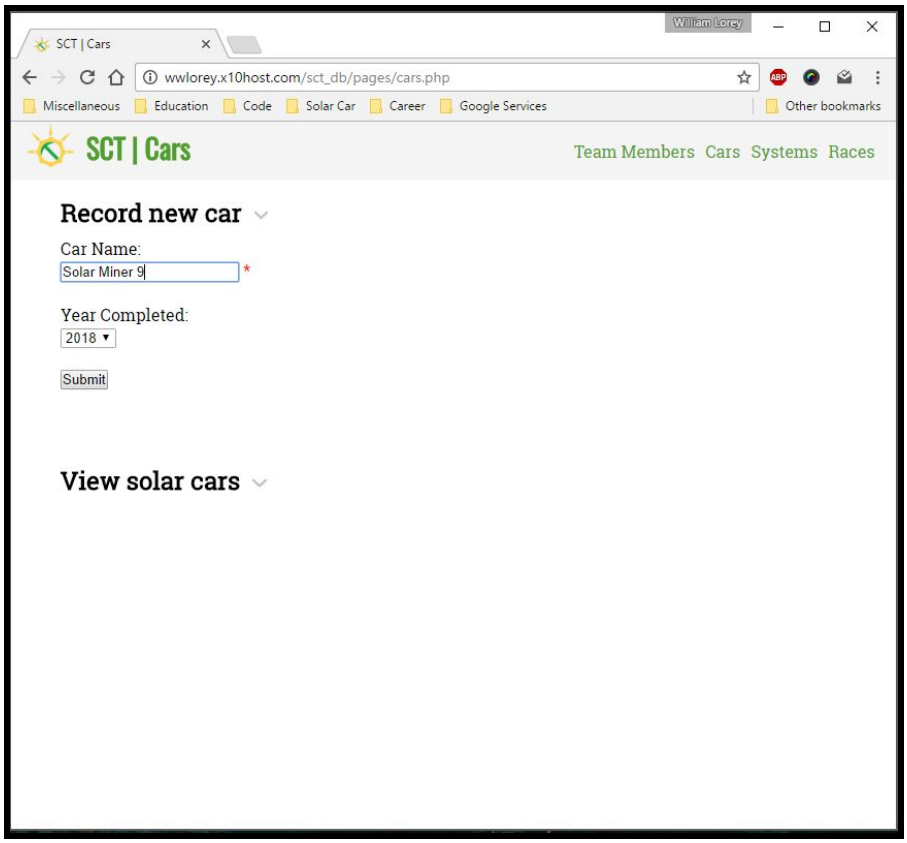
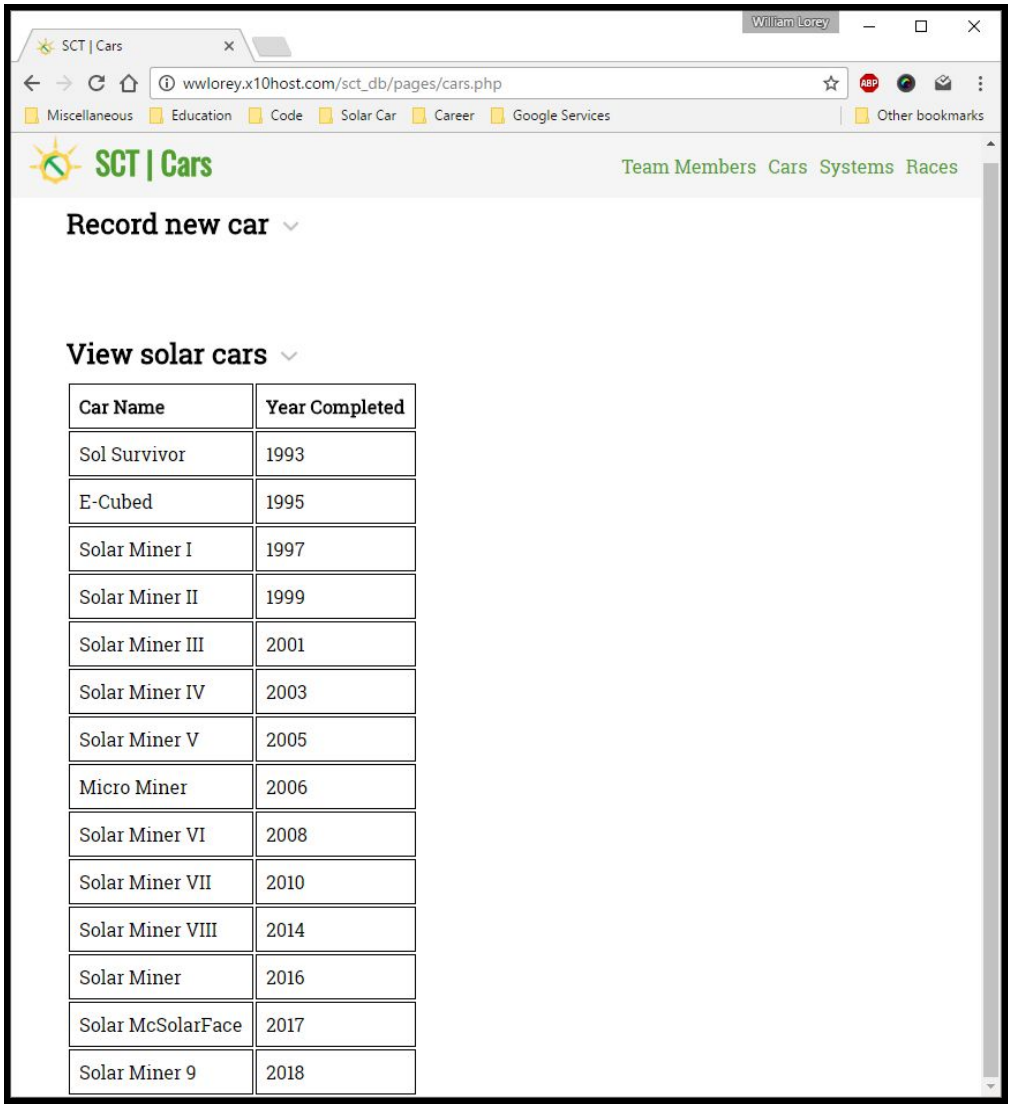


Image 6: Hiding the "Record new car" form and opening the "View solar cars" table, which has been populated to include the new car entry. Opening this tab kicks off the following query to run: "SELECT * FROM VEHICLE ORDER BY YEAR_COMPLETED". (Different views will be present such as different sorting on the vehicles, and different tables from which data is obtained).



For other DB entities (relations) encompassed by the application, similar processes will be followed to perform similar tasks to the example above (such as queries, updates, etc.). For example, a typical scenario of user interaction with the application would be viewing which team members drive which car. To accomplish this, they would click on the car name from the "View solar cars" table on the "Cars" tab to display more information about that particular vehicle, including the car driver (which comes from the DRIVE table in the mySQL database). Another typical scenario of interaction with the application would be from viewing which systems team members are working on. This could be viewed by navigating to the "Systems" tab and clicking the particular system you are interested in, which would display all team members working on that system. This interaction would make use of the WORKS_ON table.

Implementation plan

Figure 1: A Gantt Chart outlining the schedule and assignments.

First, the get user input module is tested by entering test data and validating the data. That data validation must be tested, or else we could end up with errors in our database updates and retrievals. Next, a connection to the database will be tested. This verification is required so that adding a record, removing a record, along with the SQL statement preparation can effectively communicate with the database. The add record and remove record modules will be tested following the connection. As with the first module, test data will be inserted into a table and the new record will be verified. Next, that record will be removed and we will check for a clean removal from the database. All other queries and updates are finalized in the prepare SQL statement module, ensuring they are all proper SQL statements. Finally, we verify the display data module by combining the above steps and checking for the expected results. This requires the graphical user interface to be properly functioning. Following the completion of all of this testing, the application as a whole is tested by running through typical user interaction for any anomalies that might occur. Once this is finished, the application is ready for submission.

Code listing

The following is a link to the project github page. All pertinent project files and documents are hosted here.

https://github.com/wwlorey/SCT_Electrical_DB

Sample output

Image 7: The welcome page. From the welcome page, the user can navigate to the shown tabs to perform various interactions with the database.

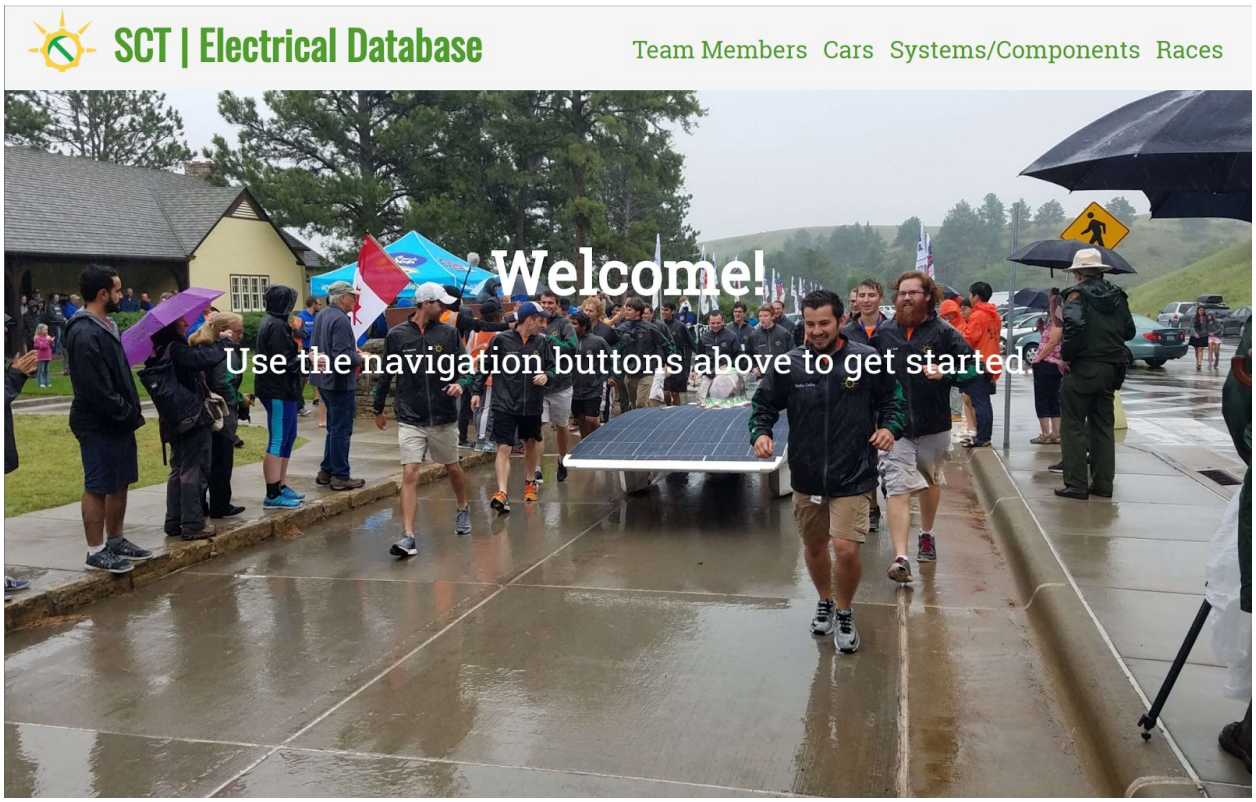


Image 8: Opening the "Cars" tab from the navigation buttons in the upper right corner of the window.

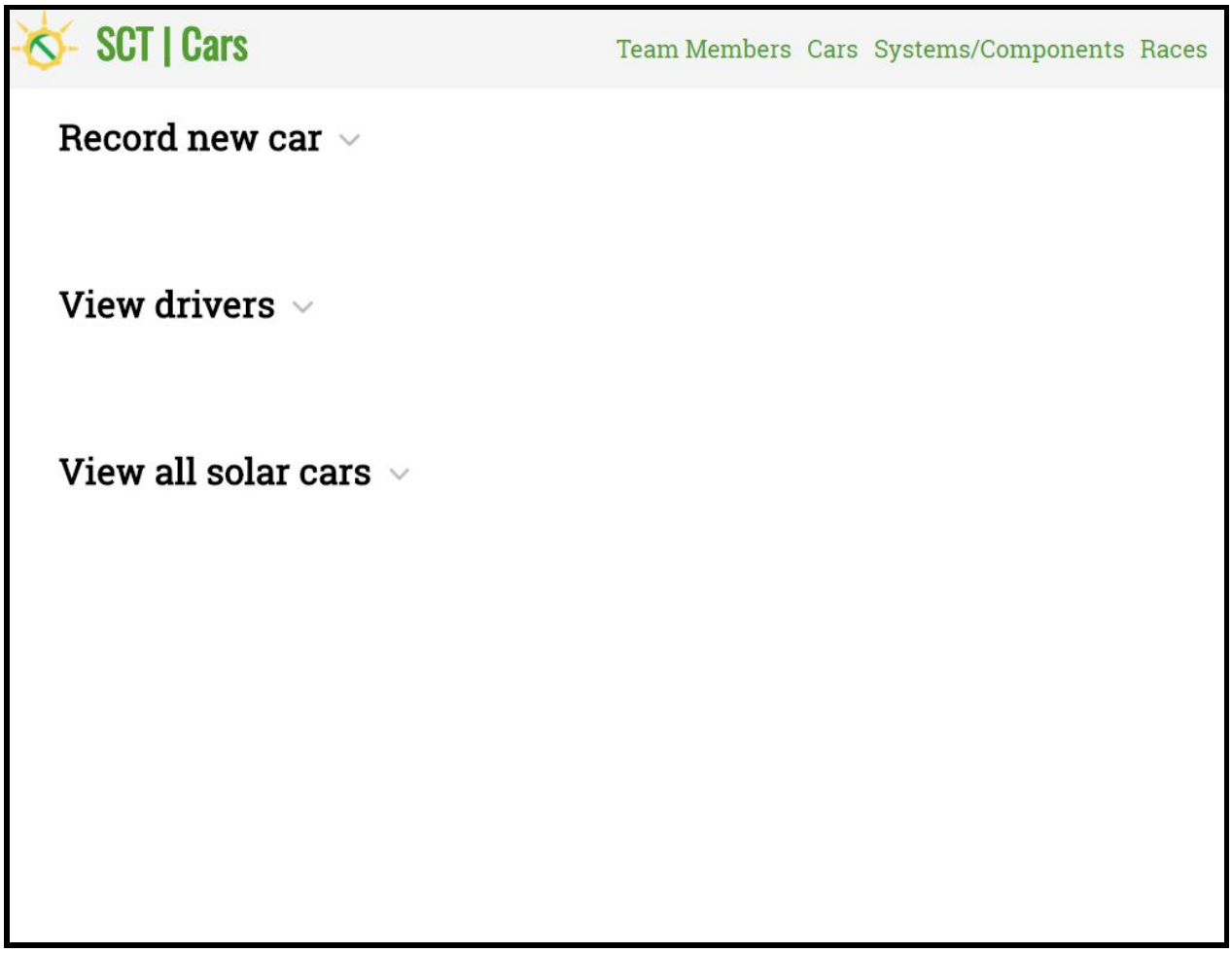


Image 9: Opening the "Record new car" dropdown form.

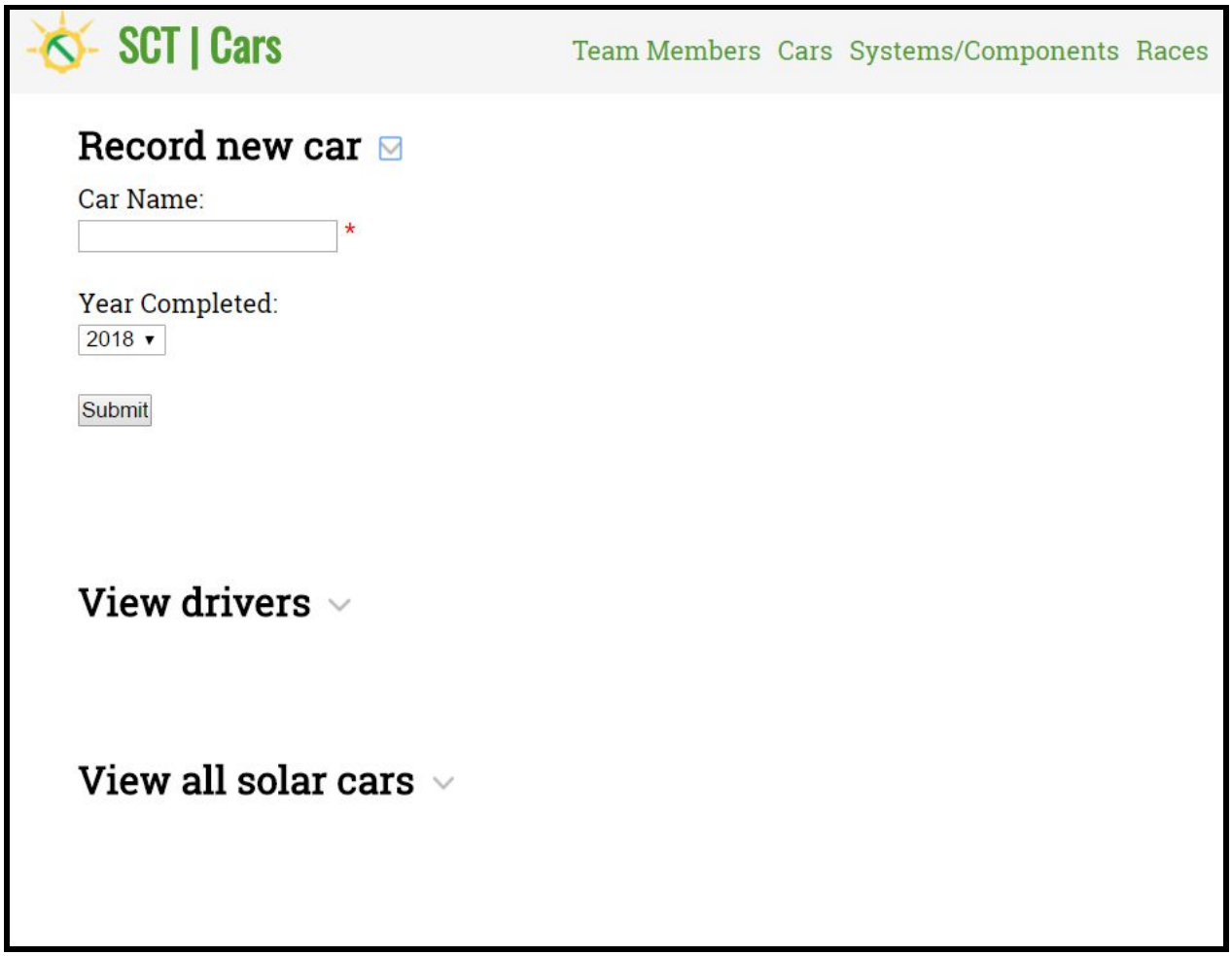


Image 10: Demonstration of table constraints: When the "Submit" button is pressed without a "Car Name", the submission is blocked. The application does not allow a user to submit a car without a "Car Name" (the NAME attribute in the VEHICLE table).

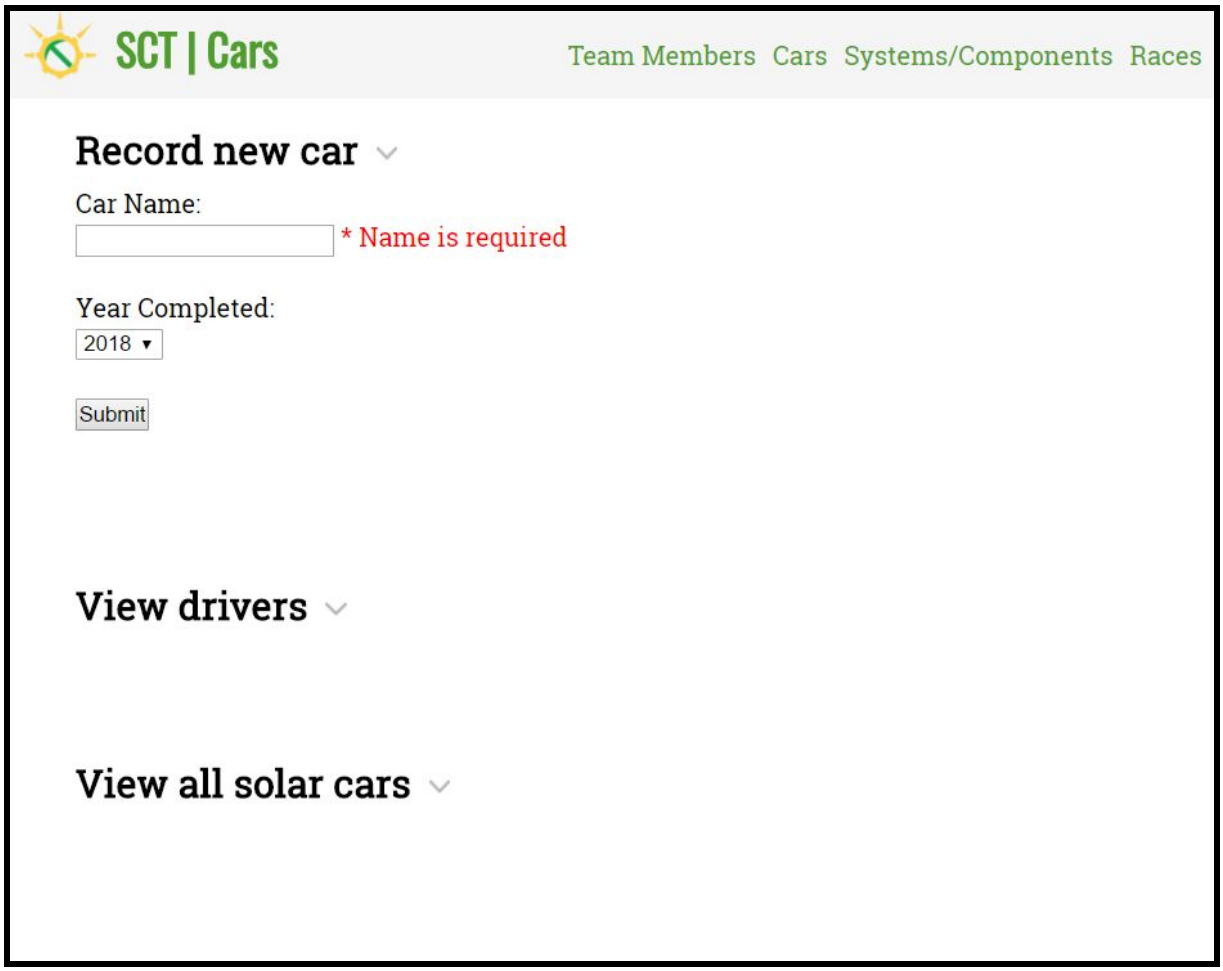


Image 11: Recording a new car and supplying a car name to the form.

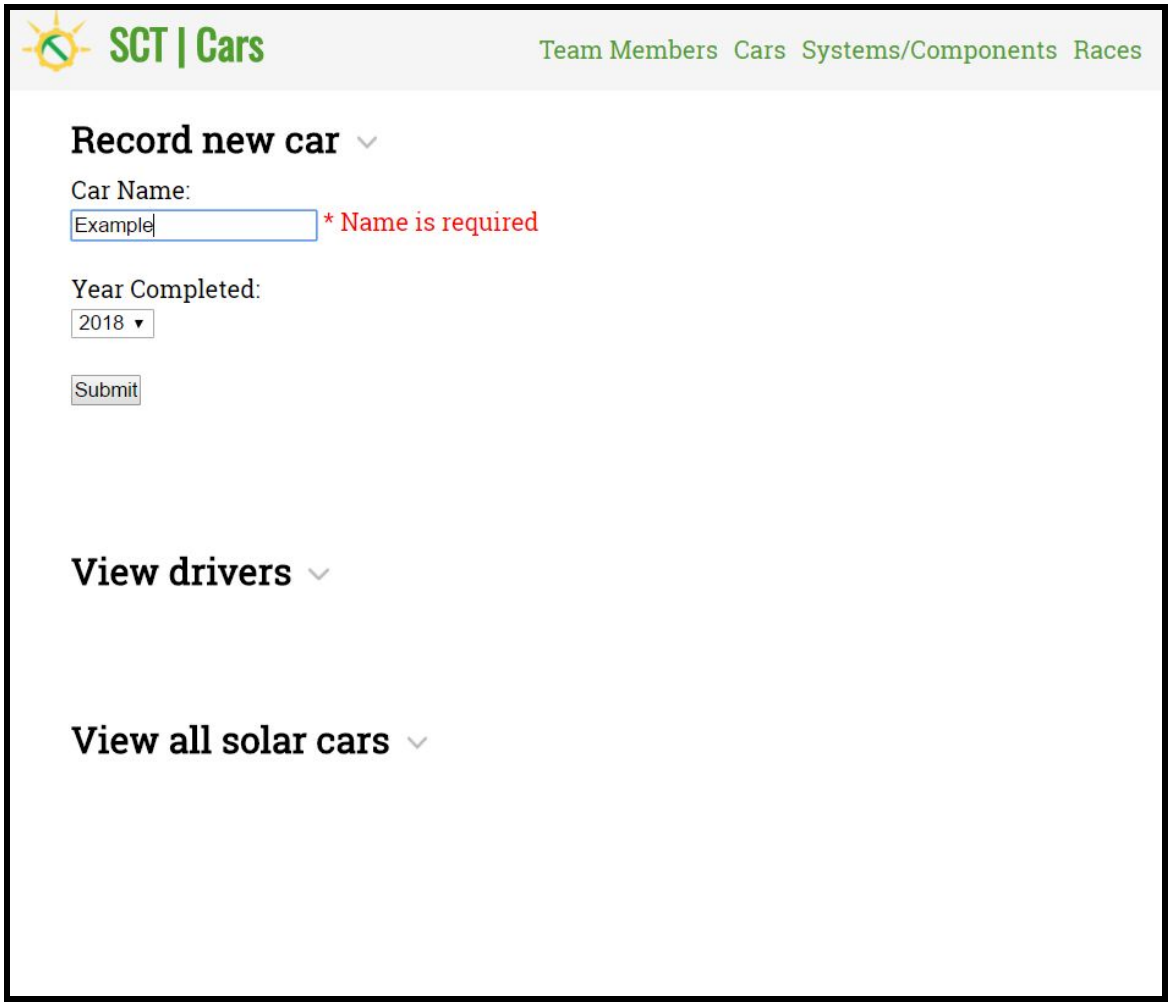
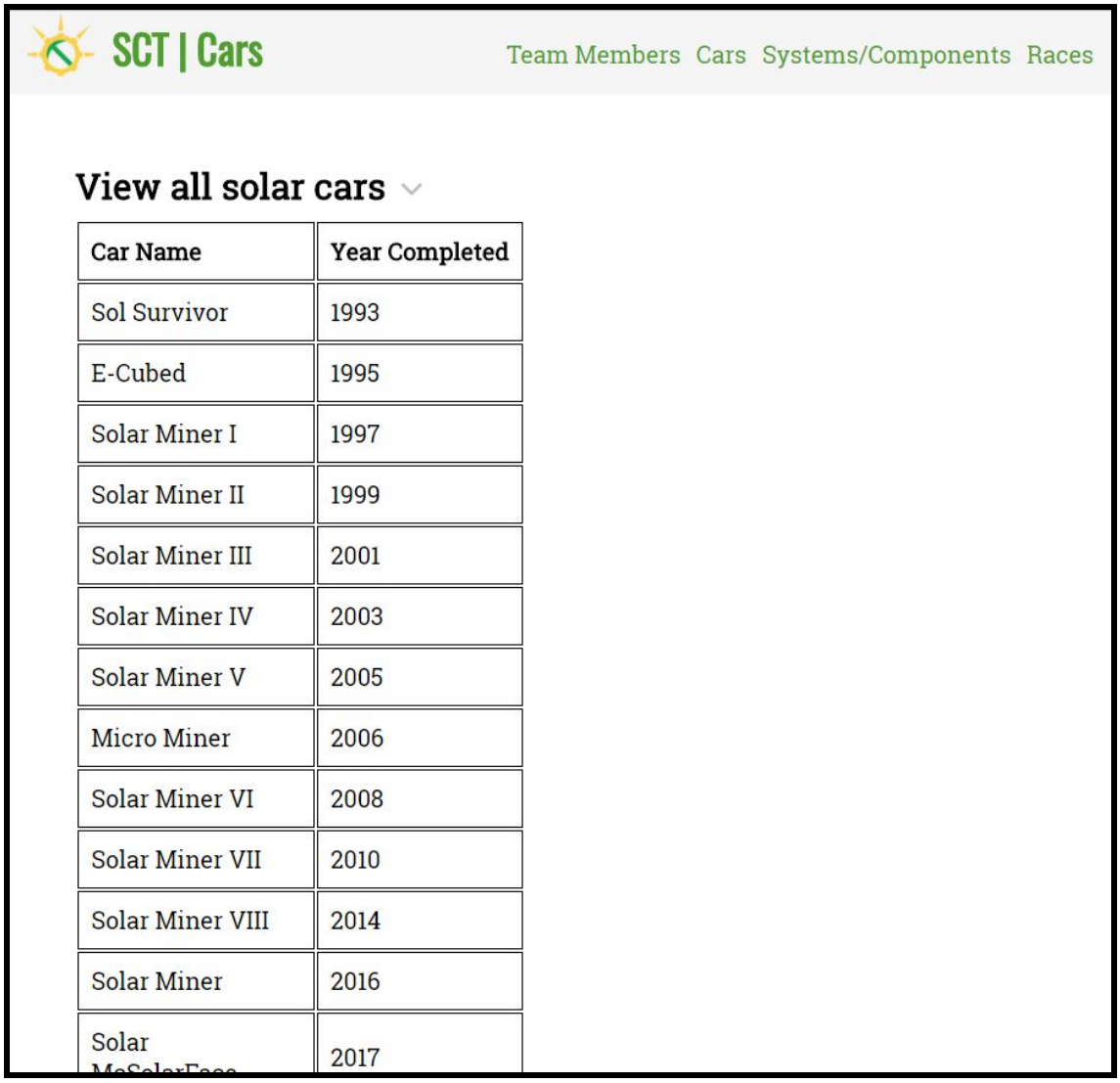


Image 12: Hiding the "Record new car" form and opening the "View solar cars" table, which has been populated to include the new car entry. Opening this tab kicks off the following query to run: "SELECT * FROM VEHICLE ORDER BY YEAR_COMPLETED". (Different views will be present such as different sorting on the vehicles, and different tables from which data is obtained).



Car Name	Year Completed
Sol Survivor	1993
E-Cubed	1995
Solar Miner I	1997
Solar Miner II	1999
Solar Miner III	2001
Solar Miner IV	2003
Solar Miner V	2005
Micro Miner	2006
Solar Miner VI	2008
Solar Miner VII	2010
Solar Miner VIII	2014
Solar Miner	2016
Solar	2017