

Team Orange (Delta V Innovation Database Team)

**Database Coding Assignment**

CS499-001 – Spring 2019

4/12/2019

**Team Members**:

Jonathan Cano

Jacob Huff

John Meyers

**Customer**:

Mike Flamm

President of Delta V Innovations Inc.

deltavinnovations@gmail.com

1. Implementation

**1.A.1 Source Code Listing**

Each team on the Delta V Innovations Projects shares a GitHub repository for code storage and access. Our group’s code on the Github/team website is small and focuses on encryption and hash testing with the database because the login and inserting features for the app and UI have not been implemented yet. As such a json file is used to simulate getting user inputs for testing purposes.

<https://jacobhuff.github.io/team4/>

<https://github.com/mfp426/SeniorDesign499>

**1.A.2 SQL Queries**

Our primary task was to merge the Vehicle Specs and Vehicle specs Additional tables to reduce the number of searches and do any cleanup work necessary.

Things to Note:Vehicle Specs has model\_make,model\_name,model\_trim,model\_year

Vehicle specs additional has make, model (which may also have what would be considered model trim in it or it could just have model\_name) and year.

1. We found multiple duplicates (~9,000) in the Vehicle Additional table

SQL QUERY—Create Distinct Vehicle Additional as (Select distinct \* from Vehicle Specs Additional)

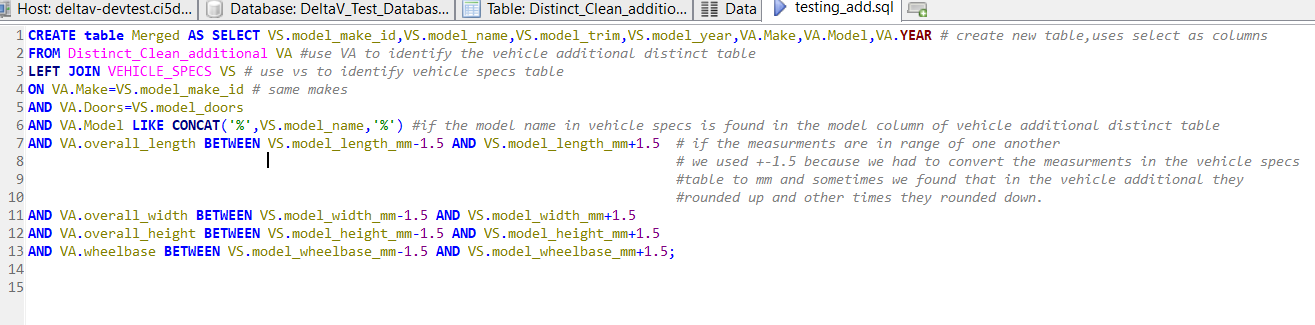
Selects only distinct rows in Vehicle Specs and inserted them into a new table

1. Found cases (~2,500) where the model make, name and year were the same in the Vehicle Additional Table, which creates an issue because we use those columns to search in the App and UI. The difference in these rows was the curb weight in most cases. The issue was that csv files each had a labeled year, but they contained info on cars made years prior to the one on the file, resulting in some cars appearing 2 or 3 times in the database. We could find no explanation for why the curb weights were different and every other column was typically the same.

Solved by- Reimporting the csv files in order from 1971-2018 and have an auto incrementing id column. We then group them by make, name, trim, and year and then take the row with the max id and insert it into a new table. So, if there was only one of that car in the table in would be inserted into the new table, but if there was 4 of the same car, the latest one entered would be inserted into the new table

SQL QUERY- create table Distinct clean additional as select max (ID), all the other columns GROUP BY make, model, year;

1. Merging the two tables based on several columns and creating a table to store those that we found could be merged together. The query below was our largest merge(~1000 merges)



From the new table created we can now identify which rows need to be merged based on the make, model, and year in the merge table and those that need to be inserted (those rows not in the table). We can also keep track of what was merged and inserted to verify accuracy. We also did some minor merges to account for columns having missing information (such as the doors column).

**1.A.3 Quality Review**

For our SQL queries we wanted them to be able to run as efficiently as possible and none that were updating or inserted were carried out on the actual table until tested on copies to verify accuracy. We had some issues with queries running over 5-6 min tended to crash HeidiSQl, though this seems to be more of an issue with HeidiSQL and not our queries based on online research. And after 1 or two attempts it would seem to work fine. For the php encryption/hashing files we wanted to write them with plenty of comments as these will serve as a baseline for future encryption project. As well as implement algorithms that haven’t depreciated or been broken. And have it so even when users enter the same inputs they don’t show up as the same encrypted string in the database.

**1.B User Manual**

The mobile app is available on the Apple App Store and Google Play Store. It is called “Delta V Field Lite”. Internet access is required for use. A user does not need to login to use the app currently. At its current iteration it only has reference material. It has been updated to use the vehicle specs merged table. The UI being built with Qt is not currently available for public use.

**1.C Administrators Manual**

The administrator’s manual can be found on the team website. It explains the process of connecting to the database through HeidiSql and the best practices we found for importing new data and testing with the database. It also explains how to connect to the AWS file server where the php files used by the app are stored.

2. Testing

**2.A Test Plan**

Over the span of the last few weeks, plenty of precautionary measures and a lot of testing has been done to ensure our methods are correct, and the database is serving its purpose. When updating and maintaining a database, it is important to create structural tests, functional tests, and nonfunctional tests.

The structural database tests deal with data integrity and non-redundancy. The primary goal of our team has been to successfully merge two large preexisting tables within the database. This kind of testing is vital to ensure the correctness of the merging process and to verify that no data has been lost or corrupted.

Functional tests were also important and used frequently. This kind of testing ensures that the actions performed by the end users is consistent and lines up with the desired output. The nonfunctional tests are the type that test the integrity and optimize the current database. When completing this project, it is important to include a mixture of all these kinds of testing methods so that the database has no potential holes or security risks.

**2.B Test Cases**

*Test Case ID:* 1

*Test Priority:* High

*Module Name:* Vehicle\_Specs duplicate rows test

*Test Designer:* Jonathan Cano

*Test Produced On:* February 28, 2019

*Summary:* A number of duplicate rows were detected in the combined table after the merge. This test is to ensure we were successful in removing the duplicates.

*Test Steps:*

1. Run query to pull data from the VEHICLE\_SPECS table in the database
2. Verify that the data contains no duplicates by inspecting the ‘inserted’ column

*Result:* Success. The current table is clear of duplicates. Approximately 9,000 duplicates were removed from the initially merged table.

*Test Case ID*: 2

*Test Priority*: High

*Module Name*: Accuracy of “Vehicle Lookup” on mobile app

*Test Designer*: Jacob Huff

*Test Produced On*: March 7, 2019

*Summary*: This test ensures that when a user enters vehicle information into the mobile app, the correct data is returned from the database. This test could only be performed once the database merge was complete.

*Test Steps*:

1. Open the mobile app and select the “Vehicle Lookup” module
2. Enter the necessary vehicle information into the fields
3. Verify that the correct data was returned to the user after searching the database

*Result*: Success. The correct data is being returned to the user from the database.

*Test Case ID*: 3

*Test Priority*: Moderate

*Module Name*: Data encryption

*Test Designer*: Jacob Huff

*Test Produced On*: April 10, 2019

*Summary*: This test ensures that we are correctly encrypting the data into the database.

*Test Steps*:

1. Run a script that encrypts and inserts user information into the database
2. Check the database to verify that the encryption worked properly

*Result*: Success. The data is correctly encrypted in the database

*Test Case ID*: 4

*Test Priority*: Moderate

*Module Name*: Data decryption

*Test Designer*: Jacob Huff

*Test Produced On*: April 10, 2019

*Summary*: This test ensures that we are correctly decrypting the data from the database.

*Test Steps*:

1. Run a script that decrypts user information stored in the database
2. The script will display the user information on a local webpage
3. Verify that the displayed data matches the stored data

*Result*: Success. The data matches and is correctly decrypted from the database.

*Test Case ID:* 5

*Test Priority:* Low

*Module Name:* Password Reset Email Verification

*Test Designer:* John Meyers

*Test Produced On:* Expected April 20, 2019

*Summary:* This test will test to make sure that when a user enters their email into the password reset webpage an email is sent to their email address with a new, functioning password.

*Test Steps:*

1. Open the password reset webpage.
2. Enter an email of a registered account and hit the password reset button.
3. Open the email account of the associated email address.
4. Verify that an email has been received with a new password.
5. Login to the Delta V Innovations website with the email address and new password to verify that the new password is functional.

*Result:* N/A

**2.C Quality Review**

We recently completed the scripts that will encrypt and decrypt certain information stored and retrieved from the database. Based off this work, we created two more test cases to ensure that these new features were working correctly. Our other test cases were created while merging tables and maintaining the database. Performing these tests routinely allows us to maintain our progress and ensure that everything is working correctly. As we add new features, we need to run these tests to make sure no previous features were altered.

3. Technical Metric Collection

**3.A Estimated Story Points**

1. **Merge two tables containing large amounts of vehicle data**

Move large amounts of vehicle data from one table into another, accounting for duplicate and overlapping data.

15 points

1. **Add new tables to the database based on what is needed from desktop team**

Enter new columns into the database in order to hold data that the desktop team will need for their implementation.

5 points

1. **Allow website access to database for password retrieval**

The website will have a password retrieval feature, and we will need to ensure that the correct data is being retrieved to reset the end user’s password.

3 points

1. **Communicate with desktop team and customer**

We will be working alongside the desktop team this semester, and it is crucial that we communicate with them in order to keep the database updated to fit their needs and keep the customer happy.

3 points

1. **Database maintenance**

It is our job to make sure the database is functioning correctly throughout the semester. This includes ensuring the data is valid and the desktop and mobile teams as well as end users are able to access and store data.

3 points

**3.B Actual Lines of Code**

As we are working with a database, we have not written much code. The only things we have written were two php scripts for encryption and decryption, totaling 110 lines.

**3.C Complexity of Each Module**

We have worked almost exclusively with three tables throughout the semester:

VEHICLE\_SPECS: 12.5 MB

VEHCILE\_SPECS\_ADDITIONAL: 4.5 MB

VEHCILE\_SPECS\_MERGE: 15.5 MB (Merged version of pervious tables)

**3.D Complexity of Overall System**

The database has a total of 29 tables that add up to 54.2 MB.

**3.E Product Size**

There are currently five user stories that total 29 story points in size. We have five acceptance test plans, two of which were created very recently following the completion of our encryption and decryption tasks.

**3.F Product Effort**

Each team member is working roughly 7 hours per week, which means that a total of 21 hours per week are being spent on the project.

**3.G Defects**

We have run into two defects, both of which did not pose a problem. The first defect was an issue where there were thousands of duplicate rows in the database. We will remove these duplicates and continue with the merging process. The second issue that we found were rows that were completely identical except for the curb weight value of the vehicle. The customer told us to take the vehicle with the lightest curb weight, and to ignore the others.

4. Developer’s Notes

We have a website hosted on github to track our progress. It contains relevant files and documents as well as developer notebooks for each team member. The site can be found here: <https://jacobhuff.github.io/team4/>

5. Demonstration

We will demonstrate our project to the class during our presentation through a live demo.

**Word Count**:

Jacob Huff: 601

Jonathan Cano: 738

John Meyers: 602