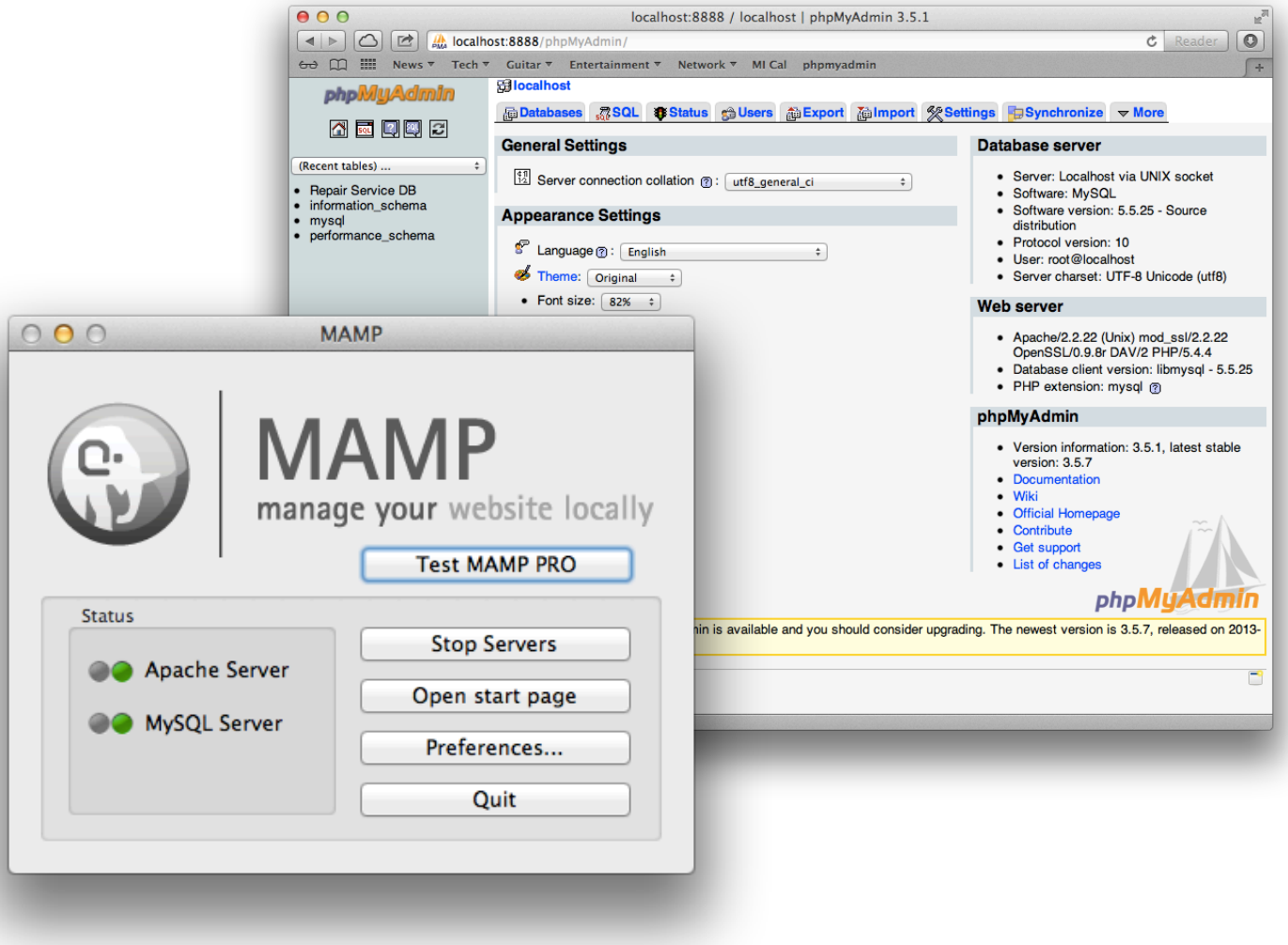
Developer Documentation - Repair Service Database

The following sections outline the process of creating, populating and exporting our Repair Service Database. Please refer back to previous documentation for information on the planning and design process of the database.

Section 1 - The Tools Used

The main tools used for this database are known as a MAMP stack: Mac, Apache, MySQL, PHP. With these tools installed we are able to create, modify and export our database using a popular interface called phpMyAdmin (Figure 1). While phpMyAdmin can handle tasks for the initial creation of the database, tables and data, its limitations meant a large portion of our work was done writing SQL queries.

Figure 1. MAMP Stack GUI and phpMyAdmin Homepage



Section 2 - Implementing the Database Design

Creating the database in phpMyAdmin is facilitated through the 'Databases' tab under 'Create Database', specifying the name for the database (Figure 2). This feature of phpMyAdmin worked well for our purposes.

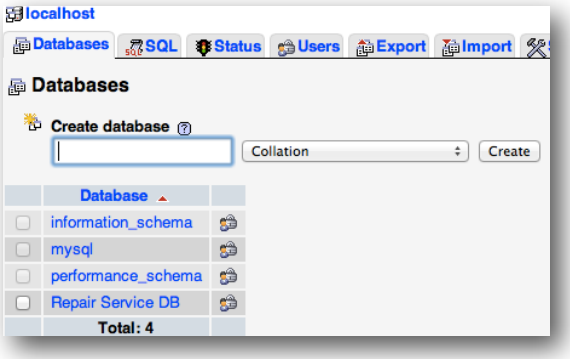


Figure 2. Create Database in phpMyAdmin

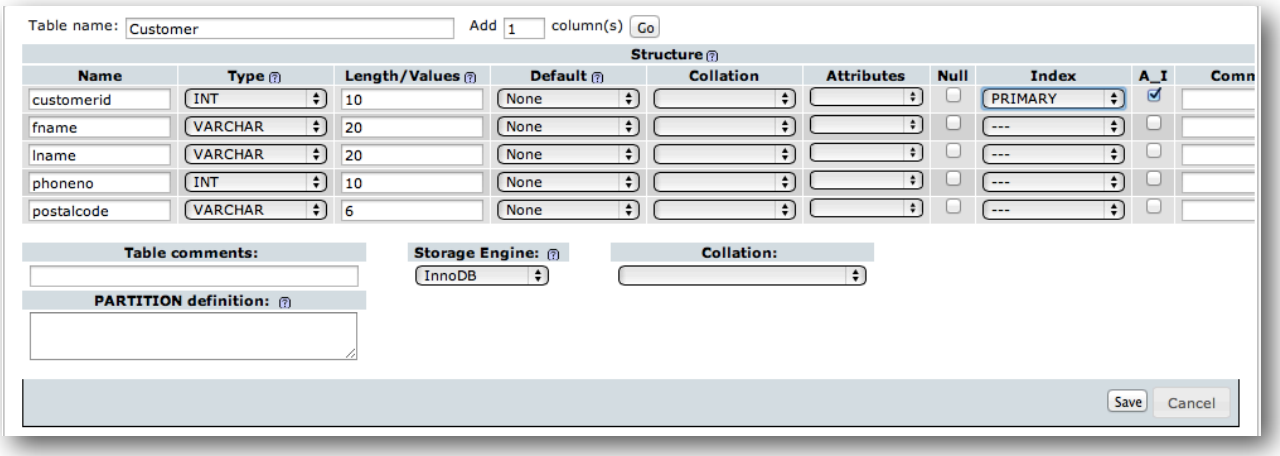
Once the database has been created and opened, we begun creating tables based on the relational model we illustrated in milestone 2 through the create table window (Figure 3).

However, our major criticism of this process is that Foreign Key constraints cannot be expressed through this interface. Additionally, conditions such as, `ON DELETE CASCADE`, require SQL statements after the table is created. For this reason, we did not use the create table window, but created our tables using SQL statements in the 'SQL' tab.

For example, the SQL statement for `Repair Notes`, a weak entity, incorporates the primary keys from `Employee` and `Repairs`. Using this method we can specify `ON DELETE CASCADE` and `ON UPDATE NO ACTION`.

```
CREATE TABLE `Repair Notes` (
title VARCHAR(100),
created timestamp ,
body text,
techid int(6),
repairid int(7),
PRIMARY KEY (title, created, repairid),
FOREIGN KEY (techid) REFERENCES Employee (techid),
FOREIGN KEY (repairid) REFERENCES Repairs (repairid)
      ON DELETE CASCADE
      ON UPDATE NO ACTION )
```

Figure 3. Create Customer Table in phpMyAdmin



Section 3 - Populating the Database

Once the structure of the database has been completed, including all of the foreign key constraints, populating the database is relatively easy. Since, this database is new, there is no existing dataset to be imported (though this can be accomplished through the import tab using a variety of formats).

After selecting a relation (created in section 2), the 'Insert' tab in phpMyAdmin allows for easy entry of new tuples. The best part of this interface was that foreign key attributes are automatically populated with values from other relation tuples. (Seen in Figure 4)

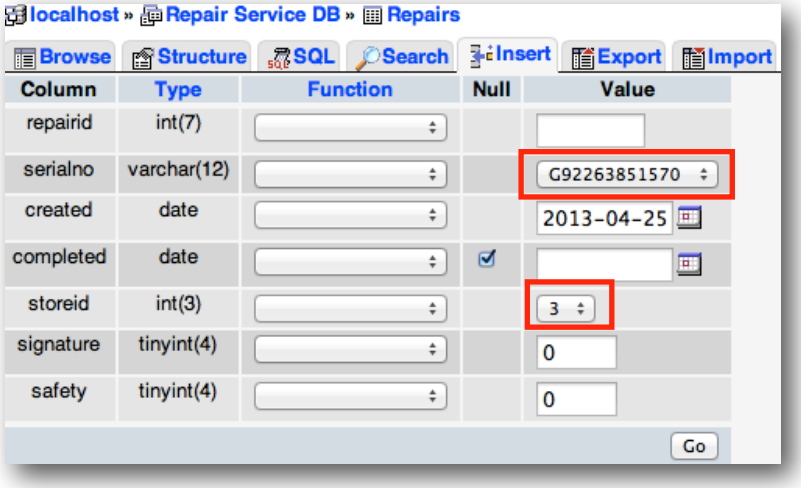


Figure 4. Insert tuple into Repairs table, with Foreign Key drop down menu highlighted

The attributes that were set to `AUTO_INCREMENT` when the tables were created in section 2 do not require a value at this stage, but will simply automatically insert a value based on the last tuple added. This is the case for some primary keys such as, `Customer.customerid`, `Repairs.repairid`, and `Store.storeid`

By the end of this process we had completed our working prototype for the Repair Service Database. Figure 5 shows the resulting 10 tables (8 regular tables and 2 views) totaling a whopping 288KB. In real world situations, this should be expected to be much larger in size as the number of tuples per table grows to the hundreds of millions.

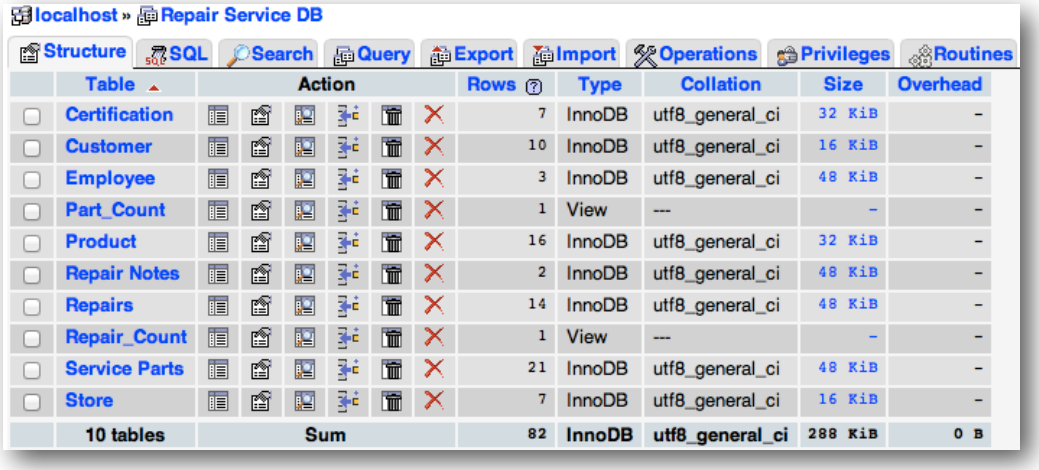


Figure 5. Overview of the Database in phpMyAdmin

Section 4 - Exporting the Database

Exporting the Database was done through phpMyAdmin's 'Export' tab. Selecting the 'Export' tab from the 'localhost' section of phpMyAdmin, will ensure that you are exporting the entire Database. Since we need to specify additional criteria for our output we selected the customer export option (Figure 6).

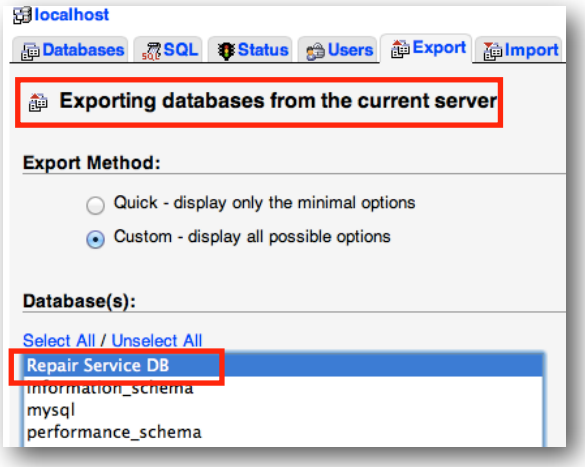


Figure 6. Custom Export of Repair Service Database

In order to output the Database as single .sql file we need to switch the default 'View output as text' to 'Save output to a file'. (Figure 7)

Under 'Data dump options' we need to specify for the output format for `INSERT` statements. To make it easier to read, we decided to output table columns and tuple values. Lastly, we chose to deselect 'Dump binary columns..' to keep output small in size. (Figure 8)

When we run the output phpMyAdmin produces a .sql file that can be easily imported into another MySQL server.

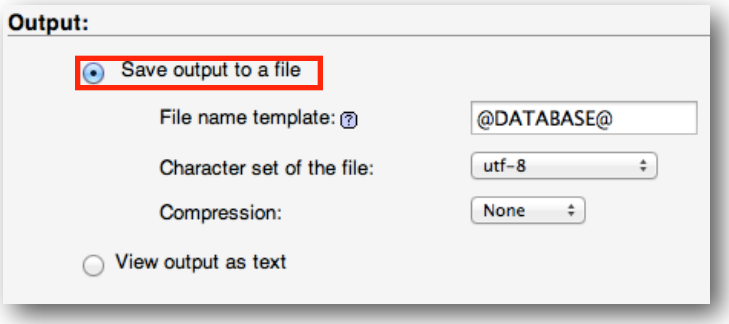


Figure 7. Save to file output option

Summary

The processes laid out in the previous sections demonstrate how to create a database using phpMyAdmin in a MAMP stack environment. However, it is important to emphasize how critical it was to have a clear, well thought out relational model and ER diagram to reference from. Moreover, in order to create tables that meet all of the constraints outlined in the design stage, a solid understanding of SQL is an absolute necessity.

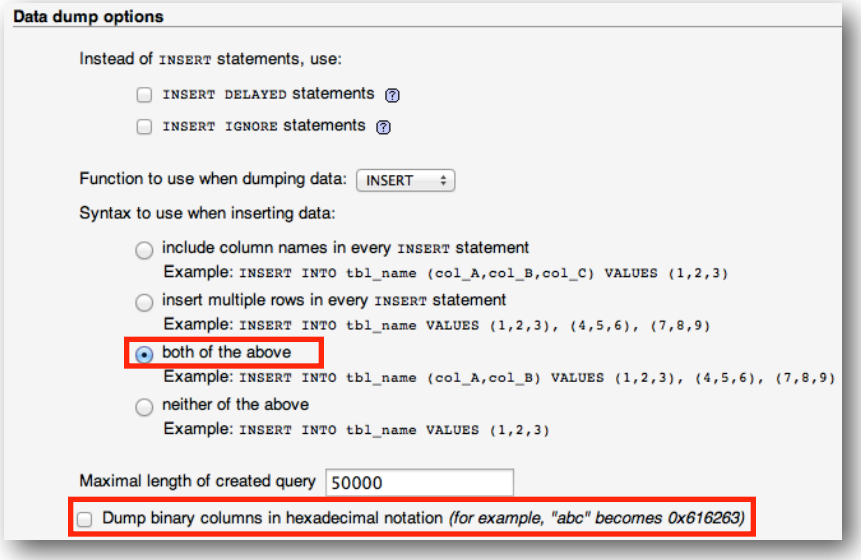


Figure 8. Data dump options