Teaching Case

Relational Database Modeling and Implementation – Food Truck Business Case

Keri Larson
larsonk@wlu.edu
Business Administration Department
Washington and Lee University
Lexington, VA 244501 USA

Abstract

After being closed for a year due to Covid, a local food truck business has engaged your tech group to design and implement a database backend for their organization's revitalization efforts. Part of your team has spent a week with the company gathering requirements; now it is your turn to plan and implement a system based on these needs. The case focuses equally upon 1) the high-fidelity design of a database capable supporting a small local business's goals of achieving profitability while serving their community, 2) coding and querying, and 3) understanding data dependencies (e.g., students experience what it means that they cannot add a dependent record to a "many" table that doesn't already exist in the parent "one" table, even when batch uploading data). This scenario presents a blend of business and technology concerns, making it appropriate as a data management project in a business analytics undergraduate program. However, given its general appeal and accessibility to students, previous business administration coursework experience is not required. As a capstone assignment in an introductory database management course, this project requires course material coverage at least through many-to-many relationships prior to commencement. Multiple suggested milestone assignments include the development of a preliminary data modeling diagram presented in a consultation meeting with the instructor, implementation of the model on a MySQL server, population of tables in a sequentially determined order, development of non-trivial SQL queries with business justifications, and the presentation of a "query matrix."

Keywords: database design, teaching case, SQL, data modeling, MySQL Workbench, team project

1. INTRODUCTION

Your student-run community-outreach tech group, "The *Codd*ifiers," has been hired by Food for Thought, LLC (FFT), the parent company that runs four food trucks in the Shenandoah area: Lex Mex, Mac Mart, Oink-Moo BBQ, and Mr. G's Donuts. As FFT looks to ramp up business after a year of closure due to Covid, its Food Truck manager, Evan Huntley, has determined that FFT requires a comprehensive database backend to support enhanced organizational functionality. The features identified by Evan Huntley as critical to a robust business include (but are not limited to): menu creation, employee scheduling, truck location planning, inventory management, and multiple HR activities. Although The Coddifiers work pro bono, the group's capable front-line associates have nonetheless diligently shadowed FFT for a week and thoroughly questioned its multiple employees about past processes as well as future goals for how the food trucks should operate. Now that your customer-facing colleagues have compiled an exhaustive list of requirements for the new database backend, it is your team's turn to model and create the system. Once you and your programming squad have created the database backend, FFT will hire an external app developer to create the front-end systems, so your SQL script should be well-documented with comments and your tables and

columns should be named according to a user-friendly convention.

2. BUSINESS PARAMETERS

FFT ultimately plans to digitize the following business processes in phases. Based on the subset of functionalities you choose to tackle as Phase 1, develop a data model containing all tables, attributes, and relationships necessary to store the relevant data. It is not suggested that you attempt to model all of these functionalities, but that you carve out a segment that make sense to you as a coherent database of related tables.

1. Employee scheduling
Should enable FFT to schedule employees to work on certain trucks during prescheduled shifts.

2. Truck location planning
There are a few permitted locations throughout town where FFT has determined the best foot traffic is available. The database should be able to store this information granularly on a daily basis for each truck.

3. Menu creation
FFT needs the capacity to create and store menus for each truck's cuisine. These will be available to customers via an app.

4. Inventory management/ordering
Each truck requires different ingredients for different dishes, but FFT gets the best price by ordering at scale and storing at a central facility. The database should support the management and distribution of inventory items to trucks.

5. Customer loyalty program
FFT plans to implement a customer loyalty program. The database needs to support this functionality.

6. Transaction/POS reporting
Customers typically pay with a credit card, although some pay with cash. Records of all transactions need to be recorded.

7. Online ordering
Some customers order ahead via an app then pick up their orders at the truck. FFT needs to record these transactions.

8. Truck maintenance recording/scheduling
Trucks often need servicing. These records must be recorded for tax and insurance purposes.

9. HR – employee pay/taxes
Employee pay stubs (with tax withholding) must be electronically generated and stored.

10. HR – employee training
Employees are required to complete training in a variety of areas as they progress up the corporate ladder within FFT including cooking, driving, operating equipment, workplace safety, sexual harassment, implicit bias training, etc. Completion dates must be recorded.

Teams should use their discretion as to which of these functionalities they should include in their data model. There are certainly additional functionalities that teams could identify, as well, which should be included in the preliminary data model presented in conference with the instructor.

3. PROJECT MILESTONES

Milestones are intended to keep teams on track. The following deliverables must be submitted, one per team, according to the milestone dates provided:

Milestone 1: A preliminary data model must be submitted to the instructor. This will be the basis for a team consultation meeting. This version of your team's data model should be hand-drawn to enable easy revision prior to being built out in Workbench. While this is a first draft, your model should be fully formed (i.e., provides all attributes explicitly), containing all necessary tables and relationships using standard crow's-foot notation as learned in class.

Milestone 2: Each team will participate in a 15-minute meeting with the instructor to review your data model. Two class periods will be dedicated to project consultation meetings; teams will use the time not in conference with the professor to collaborate and make additional progress.

Milestone 3: Once all tables, attributes, and relationships have been finalized on paper, one team member will build the data model locally (i.e., on that person's laptop) in Workbench and forward engineer the model to the team's schema on the MySQL server. At that point, all empty tables will exist on the server and will be editable by all members of the team. If you encounter an error during the forward engineering phase, attempt to troubleshoot the problem as a team by Googling the error message before contacting the professor for help. If you encounter an error, someone else has almost certainly already encountered that error (and asked the Internet for help solving it!).

Milestone 4: Creating data for testing and illustrating queries is an important step in the process. To this end, you will populate your tables with fictitious data. Team members can work on this task simultaneously (such is the beauty of a relational database!). You can edit a table directly within the Workbench interface or you can compile data in a spreadsheet and upload it through Workbench. See "Instructions for importing CSV data using MySQL Workbench" in the Appendix of this document for instructions. As a rule of thumb, 50 to 100 records are recommended for associative tables and 5 to 30 records for independent (parent) tables. You may wish to split this task up among team members but remember that **parent tables must be populated before associative tables** and **any foreign key value must correspond to an existing primary key value in the parent table**. Use your discretion to ensure you have enough variation and content for your SQL queries to produce interesting and varied results. Data should be as realistic as possible—if every item in your database costs the same or if each customer has placed the exact same order, that will give uninteresting results and will not earn a maximum grade.

Milestone 5: Create 10 (no more, no less!) non-trivial test queries. Each query must be presented 1) in natural language (e.g., "This query averages student final grades in each section of Intro to Database," 2) with a business justification as to its purpose, (e.g., "This query enables the instructor to assess variation in performance across sections to detect any major discrepancy possibly resulting from time-of-day or class composition," 3) as SQL code along with its result set (e.g., `SELECT section, AVG(final_grade) FROM db_grades GROUP BY section;`), and 4) as one or more "Xs" in the Query Matrix (in this case, an X would go in the GROUP BY row of the matrix for this query).

Milestone 6: Submit your final deliverable as a zipped file containing: 1) your team's PDF project report and 2) your team's SQL queries in a .sql script. One submission per team.

4. DELIVERABLES

For this project, you have significant latitude as to the extent and complexity of your database. The following is a set of ***minimum*** requirements your final deliverable must fulfill in order to achieve a passing grade. The quality of your work will determine your actual grade, but failing to achieve the minimum requirements will result in an insufficient grade.

- **Data Model**: Your data model must comprise a ***minimum*** of four independent entities ***plus*** all relevant associative tables and include multiple m:m relationships. Your final grade will be commensurate, in part, with the complexity and sophistication of your data model. I will pare down any overly ambitious data model during our consultation meeting but I will not issue warnings for underwhelming models.

- **Consultation Meeting**: Teams must attend a group consultation meeting with the instructor to discuss their data model and receive feedback. All members must be in attendance and must be prepared with any questions of clarification. During this meeting teams will be advised on the order of table population, with a reminder that data must be entered into parent tables before being added to an associative table.

- **Data Dictionary**: Teams should research the components of a good data dictionary and produce one to accompany four tables: two representing independent entities and two representing associative relationships. A data dictionary is simply a human-readable document, usually in table-format, that describes each attribute in natural language, including what types or ranges of data are allowable. ***Only define four tables.***

- **Raw Data**: Teams should upload sufficient data to their tables to allow appropriate testing of your database structure and a wide variety of queries. There are no specific requirements about the number of records each table should have. A good rule of thumb is that associative tables representing relationships should have around 100 rows of meaningfully diverse data. Independent tables need between 5 - 30 records to ensure enough variety of combinations in associative table to produce interesting, varied results.

- **Queries**: Produce 10 SELECT queries. The queries should demonstrate the team's breadth of understanding of SQL (i.e., 10 simple queries will not score as well as 4 simple queries and 6 complex queries). They must address questions of realistic managerial interest (not trivial ones) as well as show appropriate use of the concepts listed

in the "query matrix" (see Appendix). Teams have the chance to perfect these queries collaboratively using all available resources (slides, in-class examples, the text, the internet, etc.). As such, queries must be pretty much perfect to receive credit—I am not lenient with extra credit in this assignment. This is a chance to demonstrate mastery of the SQL concepts covered all semester. Include these SQL queries in a script that I can execute against the team's schema along with the final report PDF.

- **Final Report**: Teams must submit a professional report that summarizes their work and provides a rationale for all design and analysis choices. The report should include the following sections:
  - A cover page with the team's name and the names of its members.
  - The data model from Workbench (File > Export > Export as PNG…).
  - A short commentary about the data model and the design choices made – max 1 page. Mention any business rules not capturable by the database that should be addressed with either a business process or a feature of the front-end app, and any assumptions made.
  - The required data dictionary.
  - 10 queries in the following format:
    a. Described in natural language
    b. With a brief explanation of the business problem each query is intended to resolve
    c. As SQL code along with the results (Query -> Execute (All or Selection) to Text), truncated if the result set is large
    d. Denoted within a query matrix (see page 4 of this document)

5. GRADING

Your pro bono work will be assessed by an independent contractor, based on the following breakdown:

1. Report – **5 points**

2. Data Modeling – **15 points**
   - Identify and record the most important business rules in the environment you are modeling.
   - Clearly define the scope of your data model.
   - Create a "high fidelity" data model that accurately represents the environment you are modeling and the business rules you have identified.

3. Data dictionary – **5 points**

4. Implement the data model by creating a MySQL schema – **5 points**
   - Create an appropriately named schema in Workbench on your local machine.
   - Forward engineer your data model onto the team schema.
   - Ensure that your implementation respects all the needed integrity constraints (properly named primary and foreign keys, etc.).

5. Generate and import appropriate test data – **5 points**

6. SELECT Queries – **60 points**
   - For each query, provide a natural language description (i.e., the question a manager would ask that would drive the query).
   - Produce the SQL code and the results of the query (truncate if the result is too long) by executing each query to text.
   - Briefly explain the business problem each query resolves (i.e., why is this query important to support your business?).
   - Produce a query matrix like the one provided on the following page indicating which concept is demonstrated by each of your queries. Multiple concepts can be showcased within a single query.
   - DO NOT submit more than 10 queries. Only the first ten will be graded.

7. SQL script – **5 points**
   - Provide commented and easy-to-read SQL queries in a .sql script as part of your final deliverable, zipped along with your project PDF.

6. ACKNOWLEDGEMENTS

This project, including the "Query Matrix" deliverable, was inspired by assignments originally created by Richard T. Watson, a Regents Professor and the J. Rex Fuqua Distinguished Chair for Internet Strategy in the Department of MIS at the University of Georgia's Terry College of Business.

Appendices

Query matrix example

	Query1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
1. Subquery	x			x			x			
2. OUTER JOIN	x			x						
3. GROUP BY			x							x
4. HAVING			x							
5. ORDER BY			x							x
6. IN/NOT IN		x				x				
7. Aggregate function(s)			x		x					
8. REGEXP			x	x		x	x			x
9. Date function(s)								x		
10. IS NULL	x	x		x						
11. DISTINCT	x								x	

Note, this query matrix reflects the concepts captured by a set of SQL queries submitted by a prior team. You are NOT expected to craft SQL queries according to this matrix, but should fill out a blank matrix that matches concepts to your own queries.

ISSN 2473-4901
v7 n5584

Instructions for importing CSV data using MySQL Workbench

MySQL workbench provides an interface for importing comma-separated values into a table. This allows you to create/edit your data in an Excel spreadsheet before uploading it to MySQL.

Before importing the file, you need to create the following infrastructure:

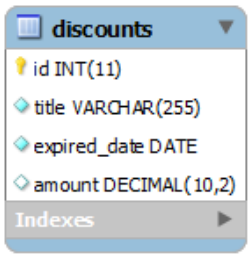
1. A database table into which the CSV data will be imported (created when you Forward Engineer your schema to the MySQL server).

2. A comma-separated-value file that you export from your Excel spreadsheet:

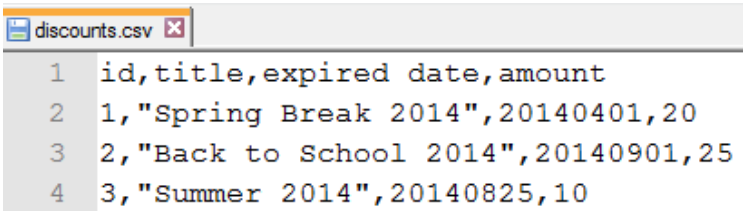
   File→Save As→ CSV file

   with data that matches the number of columns of the table and the type of data in each column.

For example, suppose you have a table called discounts with the following structure that has been Forward Engineered to the server:

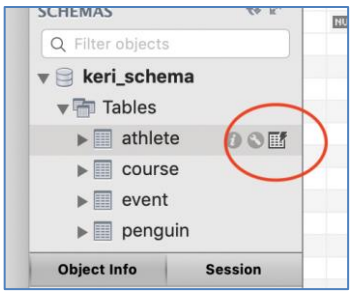


The corresponding CSV should be of the format that follows. Note that there are column names in the first line with values in the following lines.

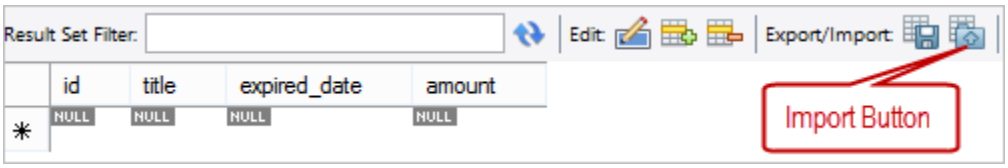


ISSN 2473-4901
v7 n5584

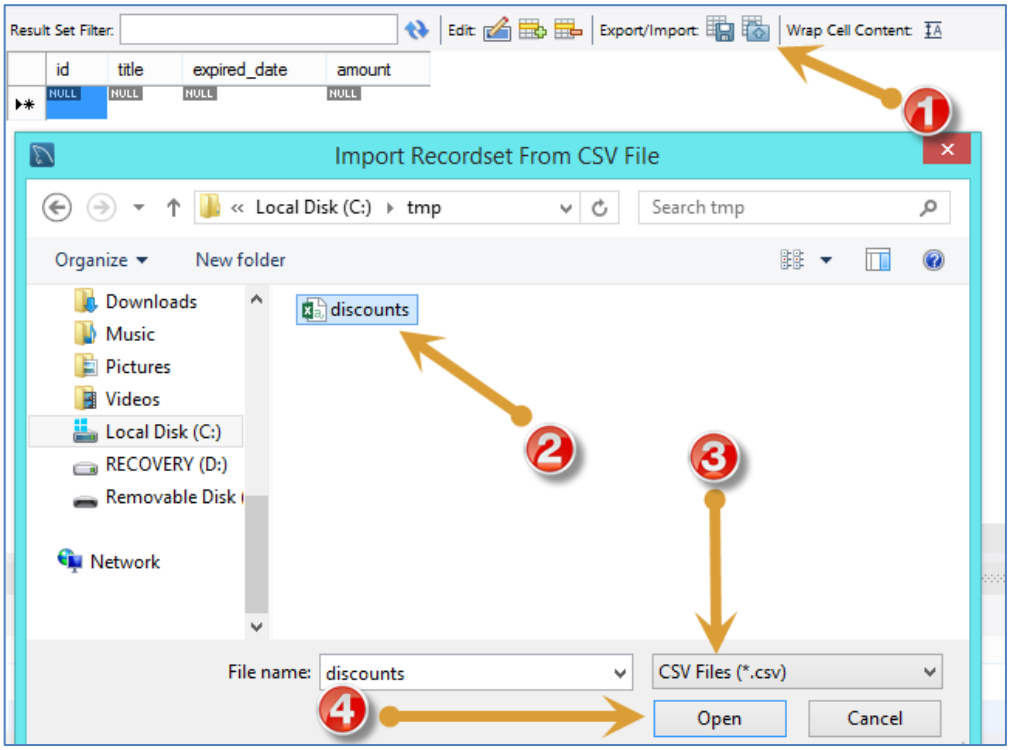
To import the data into your table in Workbench, find the table into which you want to import data in the left-side "Schemas" panel. Click the tiny spreadsheet icon next to the table:



In the new view, click the Import button:

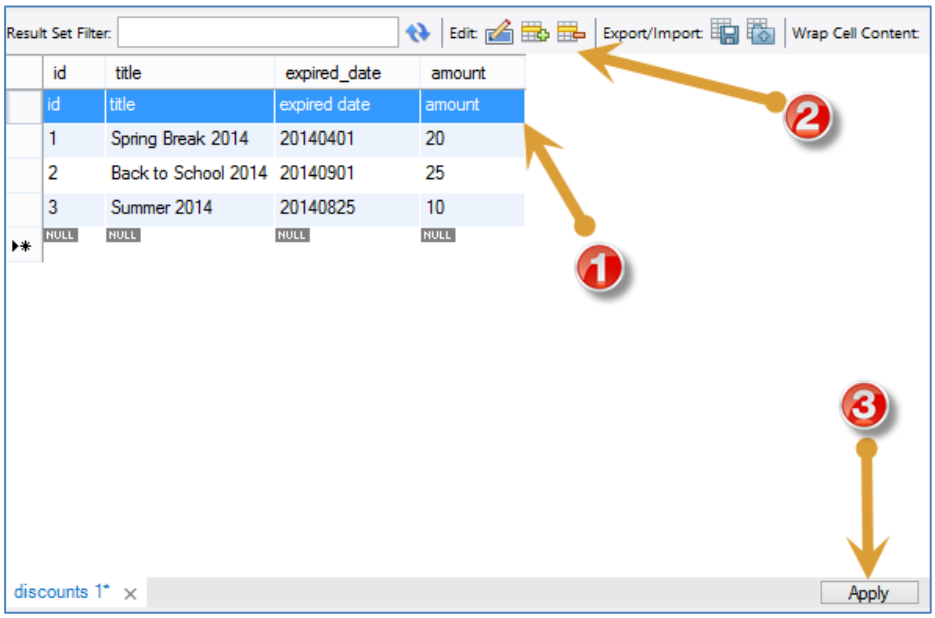


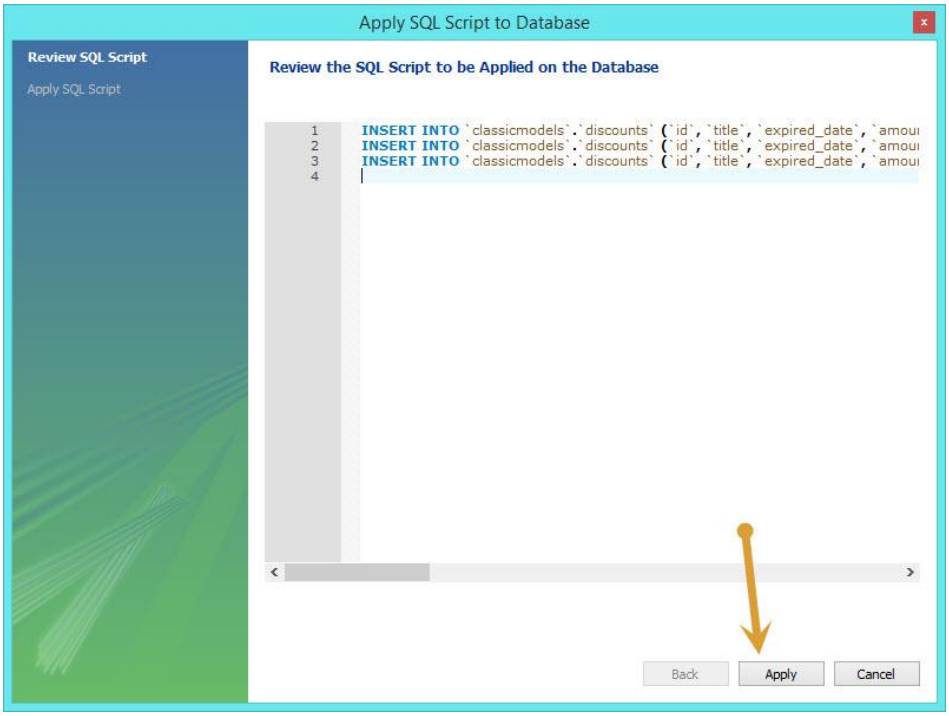
Next, click the up-arrow-folder (1) to choose a CSV file (2 & 3) and click Open (4):



ISSN 2473-4901
v7 n5584

Review the data in the preview screen and if satisfied, click Apply button. **NOTE:** you do not want to import column names into your table. If you see column names in your preview window as the first row of data, highlight the first row (1) and click the red minus-row button (2) to delete the extraneous record before hitting the Apply button.





MySQL workbench will display a dialog box entitled, "Apply SQL Script to Database." Click Apply to insert data into the table.