**Final Project: Milestone 3**

Derek Caramella, Lisa Pink, & Tapan Pradyot

Department of Computer Science, University of Rochester

CSC 461: Database Systems

Dr. Zhupa

November 15, 2021

**Task A: BCNF Normalization**

The table is in Boyce-Codd Normal Form (BCNF) when the table satisfies the third normal form properties and for each functional dependency in the relation, *X* → *Y* is a trivial functional dependecy and *X* is a superkey for the relation. The relations were in BCNF prior to a decomposition exercise; therefore, the functional dependenices are described below.

Seven tables compose the *derekcaramella$dcaramel\_1* database: (1) *Bartenders*, (2) *EmploymentTypes*, (3) *OrderableItems*, (4) *ItemSupplies*, (5) *Recipes*, (6) *Orders*, and (7) *CompletedOrders*.

Consider the relation *Bartenders*(*Bartender\_id, First\_name, Last\_name, Employment\_type*). Since first name, last name, and employment type may duplicate, for example (John, Smith, Full Time) and (John, Smith, Full Time), then the functional dependency set is {*Bartender\_id*→*First\_name*, *Bartender\_id*→*Last\_name*}.

Consider the relation *EmploymentTypes*(*Employment\_id, Employment\_type\_desc*). Since employment type description does not duplicate, then the functional dependency set is {*Employment\_id*→*Employment\_type\_desc*}.

Consider the relation *OrderableItems*(*Item\_id, Item\_name, Item\_price, Available*). Since an item name may be duplicated1, the functional dependency set is {*Item\_id→Item\_name, Item\_id→Item\_price*}.

Consider the relation *ItemSupplies*(*Item\_id, Item\_name, Item\_size, Item\_price*). Since an item name may be duplicated[[1]](#footnote-1), the functional dependency set is {*Item\_id*→*Item\_name*, *Item\_id*→*Item\_size*, *Item\_id*→*Item\_price*}.

Consider the relation *Recipes*(*Finished\_item\_id, Supply\_item\_id, Recipe\_amount*). Since no finished item and supply item with the same recipe amount is duplicated, the functional dependency set is {(*Finished\_item\_id, Supply\_item\_id*)*→Recipe\_amount*}.

Consider the relation *Orders*(*Order\_id, Order\_item, Order\_time*). Since order times prevents duplicates, the functional dependency set is {*Order\_id→Order\_item, Order\_id→Order\_time*}.

Consider the relation *CompletedOrders*(*Order\_id, Bartender\_service, Completed\_time*). Since order times prevents duplicates, the functional dependency set is {*Order\_id→Completed\_time*}.

**Task B: Create Forms Using HTML**

Sandy is a Flask application; therefore, the project contains no raw HTML or PHP files. However, embedded in the code, there are HTML divisions that render HTML tags. Applicably, in app.py, lines 116 through 416 contain HTML divisions that render graphs, slices, filters, and user forms for database interaction. The developers utilized Plotly Dash to build a python framework for the Sandy dashboard. Plotly Dash is a third-party module that wraps CSS and React in data classes for python development. The developers deployed Sandy on the Platform as a Service (PaaS), *pythonanywhere*, sever since the developers endeavored to produce a public facing web application. The application is executed on python 3.8 to utilize additional functionality and increase efficiency. The developers utilized the platform’s internal MySQL server to build the relations. The developers constructed a ssh tunneling class to ensure secure, rapid connectivity with the server to render accurate and precise results. Sandy is a python Flask application that utilizes Plotly Dash to display a dashboard that interacts with the MySQL server on *pythonanywhere*.

**Chart, diagram, box and whisker chart

Description automatically generated**

1. MySQL Callbacks with Web Application

# Task C: Create & Loading Relations

The developers were unable to acquire access to bulk upload with sql scripts to the *pythonanywhere* server. However, a load.sql script is posted for the betaweb server, which the developers utilized as a build site for the *pythonanywhere* server application. The developers employed a MySQL API, pandas library, NumPy functions, and a customized ssh tunneling class to bulk upload the comma-separated value files to *pythonanywhere* utilizing python. Moreover, the developers constructed a create.sql file to batch build the data tables and foreign key constraints in the MySQL database. Lastly, all data was synthetically generated applying customized python sampling and executions for the application.

# Task D: Accessing the Relations from Web

The website can be found with this [link](http://derekcaramella.pythonanywhere.com/) or QR code. The website is desktop and mobile compatible.

Qr code

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| Project Contributions & Responsibilities | | | |
| Contributions | Derek Caramella[[2]](#footnote-2) | Lisa Pink[[3]](#footnote-3) | Tapan Pradyot[[4]](#footnote-4) |
| * Composition Writer * Database Creation & Integration * DevOps * Software Development | * Composition Writer * Data Curator * Database Design * ER Model Developer * Quality Testing | * Composition Writer * Data Curator * ER Model Developer * Database Design * Quality Testing |

1. Although records may be duplicated, the graphical user interface (GUI) contains safeguards to prevent duplicate actions. [↑](#footnote-ref-1)
2. Derek Caramella: 32123999 [↑](#footnote-ref-2)
3. Lisa Pink: 32130478 [↑](#footnote-ref-3)
4. Tapan Pradyot: 31866309 [↑](#footnote-ref-4)