

DTSC 691 Project Submission Template

Database for Clinical Reference Laboratory

Ryan Breen

July 2nd, 2022

Table of Contents

**SECTION 1: PROPOSAL*

(follows closely to the Database Submission Template)

Background

- 1.1. Project Motivation*
- 1.2. Problem Set*
- 1.3. Implementation Strategy*
- 2. Problem Objective*
 - 2.1. Organization of Problem Objectives*
 - 2.2. Determining if Problem has Been Solved*
 - 2.3. Solutions for Ten Problems Defined*
- 3. Raw Data*
 - 3.1 Raw Data Repository*
 - 3.2. Database Tables and Columns Description (edited)*
- 4. Functional Requirements*
 - 4.1. Functional Requirements (added subsection)*
- 5. ER Diagram*
 - 5.1.ER Diagram .pdf*
- 6. Logical Schema Diagram*
 - 6.1.ER Diagram to Logical Schema*
 - 6.2. Logical Schema Schema .pdf*
- 7. Software*
- 8. Analysis*

SECTION 2: POST-PROPOSAL FOLLOW-UP

(follows closely to the Database Submission Template)

- 9. Raw Data Creation for SQL Insertion*
 - 9.1. Schema table/columns and Python*
 - 9.2. Python Function to Create SQL Insert Statements*
- 10. PostgreSQL Statements for Relational Database*
 - 10.1. Explanation for Order of Operation*
 - 10.2. Table Creation*
 - 10.2.1. Data Types*
 - 10.2.2. Constraints*
 - 10.3. Sequences*
 - 10.3.1. General Purpose and Execution*
 - 10.4. Foreign Keys*
 - 10.4.1. General Purpose and Execution*
 - 10.5. Formation of Query Tables*
 - 10.5.1. collector_containers*
 - 10.5.2. customer_reports*
 - 10.5.3. reports_GFR_table*
 - 10.5.4. reports_alert_table*

- 10.5.5. *customer_grouped_charges*
- 10.5.6. *customer_individual_charges*
- 10.5.7. *manger_grouped_qc_report*
- 10.5.8. *manager_individual_qc_report*
- 10.5.9. *manager_range_reports*
- 10.5.10. *manager_tests_report*
- 10.6. *Trigger Functions*
 - 10.6.1. *collector_containers_trig_func()*
 - 10.6.2. *customer_charges_trig_func()*
 - 10.6.3. *customer_grouped_charges_trig_func()*
 - 10.6.4. *customer_reports_trig()*
 - 10.6.5. *orders_charges_trig_func()*
 - 10.6.6. *orders_samples_trig_func()*
 - 10.6.7. *orders_tests_trig_func()*
 - 10.6.8. *Diagram of View Functions*
- 10.7. *Test Statements*
 - 10.7.1. *Trigger Function Insert and Outcomes*
- 11. *Flask-SQLAlchemy for Application Creation*
 - 11.1. *Application Considerations*
 - 11.1.1. *File Structure*
 - 11.1.2. *Dependencies*
 - 11.1.3. *Virtual Environment*
 - 11.1.4. *Local Git Repository*
 - 11.1.5. *Procfile Use and Mistakes*
 - 11.2. *Flask Object*
 - 11.2.1. *Database Reflection and Automap*
 - 11.2.2. *Data Querying and Inserts*
 - 11.2.3. *app.routes(), render_template(), Jinja templates*
- 12. *Heroku Deployment*
 - 12.1. *Heroku Database setup*
 - 12.2. *Git Push to Heroku*
- 13. *Project Website/Repository*
- 14. *Proof of Work for Trigger Function/Web Views*
- 15. *Cited References*
- APPENDIX: Source Code (In Order of Operation) (*
(this file will be added to the final submission to Brightspace)

Background

A clinical reference laboratory is a private laboratory that allows customers to submit orders online for medical laboratory tests commonly referred to as 'blood tests'. Some examples are companies such as Quest®³ or Any-Lab-Test-Now®² where customers may order lab tests online, have their blood drawn, and test results are then produced for laboratory tests that customers pay for online.

Project Motivation

While I was working at a specific *startup* reference laboratory outside of Boston, it was apparent that this company's MS-DOS version of an electronic medical record (EMR) was quite antiquated. The reasoning behind using such an older platform (Sunquest) is that for such a new company reliant solely on investments, purchasing a license for software from *giants* such as EPIC or Cerner was out of the question. This prompted me to wonder if many other small-community, startup, or university-research based laboratories could benefit from *open-source* or *low_cost* laboratory platforms?

These *laboratories* have business demands that involve transactions for finances, customer information, interplays between lab orders and analyzer middleware. Although this project doesn't come near the magnitude of demands needed by a reference laboratory, it proposes to serve as a 'low-scale' model version that hones in on several fundamental laboratory database functions that are created to *solve* the specific *problems* encountered by the laboratory enterprise. Solving these key enterprise problems should demonstrate my personal capabilities to expand career horizons in healthcare analytics and maybe provide a stepping stone for a future *low_scale* platform for any potential needs.

Problem Set

As previously described, the project motivation is to address a group of major functions that serve the demands of a laboratory by using a database design to create a 'low-scale' functional application. In this project we solve several key problems that are demands needed by each role within the laboratory. Roles include the *customer*, *employee*, and *manager*. The project uses a database, coding, and develops a web application together to serve a medium to accommodate a few specific tasks/functions. Generally speaking, several SQL procedures and functions with Python coding are created to emulate real-world transactions occurring in the laboratory for each of the roles aforementioned.

The reason for including several problems (demands) to solve versus a singular problem is that many of the problems in the laboratory are intertwined with other problems. This project aims to meet the originally stated project motivation to create a 'low-scale' model version of a platform to be used for suited needs as well as demonstrating a general knowledge of healthcare databases by the author. This requires solving several problems that have been named formally the 'problem set'. These several problems are each specified in the Problem Objectives section below, but in this section the problems are *simply* introduced as business demands needed for customers, employees, and managers of a reference laboratory.

Implementation Strategy

Prior to discussing each specific problem listed in the Problems Objectives, a very brief discussion is given to how to solve the problem set formally denoted as the Implementation Strategy. The project starts by creating raw data using Python to serve as a data source to test the various database functions that will hopefully be used to address each business demand. Raw data also serves as reference information for the web application. For example, a function that creates charges for orders should have data to test if such a function works. Also, if a 'customer' places an order on the web application, there needs to be references (via foreign key constraints) that map to other tables containing charge information and many other related items. These reasons are why it is *imperative* that the raw data is created with all the foreign keys (as columns) included in each table. In other words, this project purposely uses the logical schema as opposed to the conceptual schema to create the raw data.

Finally, raw data is directly inserted (thanks to using the logical schema) into the database. SQL DDL and DML statements are created in light of solving various

enterprise problems. The database is then linked to a web app using *Flask(Python)*. Heroku allows web applications to be hosted and users should be able to perform tasks related to *customer, manager, or the employee* standpoint; thus, fulfilling various demonstrations to solve problems .

Problem Objective

Organization of Problem Objectives

The database design has been created to meet several needs and roles of *customers, employees, and managers*. Each aforementioned role contains SQL views that are groups of related tasks performed within each role. Each SQL view is then subdivided into various SQL procedures and functions that solve problems specific to each related task. For example, a 'lab employee' is a role within a laboratory (realm) who is responsible for specific groups of tasks(jobs). One group of tasks might be collectively named 'Analyzer' and within this group exist various tasks related to the lab employee working with an analyzer. An 'Analyzer' view is created in SQL with each of the constituent tasks represented by functions and procedures that are collectively 'displayed' on an application.

Determining if a Problem has Been Solved

Outline below are each problem organized into groups that are SQL views that live within each specific role. The problems themselves are *attempted* to be solved by creating SQL views, procedures, and functions that accommodate each role. To see if each problem has been solved, demonstrations of successful queries within the SQL database are documented and submitted within the project materials. A demonstration of each procedure or function is performed on a video summary of the project to formally and legitimately document that each problem enumerated below has been addressed. The solutions should be able to work in a dynamic environment(web app).

Problems and Solutions Listed Below (organized by role and view):

Customer Role

Billing View

Problem 1: charges needed to be grouped for each individual customer with aggregated sum

Solution 1: SQL Procedure that will aggregate charges by customers with input to the procedure to filter

Test Report View

Problem 2: results need to be grouped by customer who should be able to filter by date and specific test

Solution 2: SQL Procedure that will aggregate results by customers with input to the procedure to filter

Problem 3: customers should be alerted to results that are deemed 'low' or 'high' with a relevant comment

Solution 3: SQL Procedure that aggregates results by customers with input to the procedure to filter

Employee Role

Analyzer View (for Lab Techs)

Problem 4: orders without results ('test to be run') or results with missing values should be made discoverable

Solution 4: a View table that using a procedure to seek out missing results and updates accordingly

Problem 5: calculated test reliant on patient demographics (i.e.age) should automatically calculate and insert

Solution 5: a SQL function that incorporates test and demographic data and inserts into results table

Collection View (for Lab Collectors)

Problem 6: lab collectors need to know what specific type of tubes are needed for each type of testing ordered

Solution 6: SQL procedure that maps order information with collection tube information and creates a view

Problem 7: a barcode (QR code) should created for each order per each customer

Solution 7: incorporating in a Python library that creates QR code to correspond with customer information

Manager Role

QC View (For Lab Managers):

Problem 8: QC results* that our out of range or violated specifically programmed rules must create alerts

Solution 8: Procedure that incorporates QC rules and ranges and creates subsequent commentary

Problem 9: QC results should populate by test or date unto a visualization called a Levy Jennings chart

Problem9: Create SQL procedure that interacts with Seaborn to create detailed professional visualization

Patient Ranges View(For Lab Managers):

Problem 10: Cumulative statistics needed for each customer test to aid in creating reference ranges*

Solution 10:

- QC is reagents with know test value ranges that are run to ensure analyzer accuracy
- Reference ranges are ranges customer results must fall in to be considered a normal test result

Raw Data

A Jupyter notebook and an accompanying data dictionary describing each attributes in-depth is posted privately on GitHub(invitations may be sent at any time for access): <https://github.com/RyansStacks/Capstone-Database-Project-DTSC691->

Brief description of each table and subsequent columns (logical schema)

Note, after creating the logical schema and applying normalization(i.e.3NF) some changes were made to the original raw data to reduce redundancy. Again, the raw data for this project is formulated to fit the logical schema so if the logical schema is improved upon then the raw data must be reformulated (luckily this is easy to do with Python). Also, the discovery of Postgresql (psql) functions now() for dates/times and serial data types/auto incrementation used for many primary keys in the trigger functions in this project prompted me to update these data types as well in the raw data. Changes are specifically highlighted and annotated below.

tables	columns	dtype	rows	cols	nulls	notes
customers	customer_id	integer*	500	7	0	change from string integers to integer type to utilize SQL autosequence function.
	address_id	integer*	500	7	0	
	phone_id	integer*	500	7	0	
	email_id	integer*	500	7	0	
	first_name	string	500	7	0	
	last_name	string	500	7	0	
	date_of_birth	date**	500	7	0	
orders	order_id	integer*	1000	5	0	
	panel_id	integer*	1000	5	0	
	customer_id	integer*	1000	5	0	
	order_date	date**	1000	5	0	
	order_time	time**	1000	5	0	
customer_tests	test_id	integer*	3821	7	0	merged customer_results table with customer_tests to reduce data redundancy in database. Gained results, result_date, and result_time Lost result_id FK to customer_results talbe. Hence 3 -1= 2 extra columns.
	employee_id	integer*	3821	7	0	
	analyste_id	integer*	3821	7	0	
	order_id	integer*	3821	7	0	
	result	numeric	3821	7	0	
	result_date	date	3821	7	0	
	result_time	time	3821	7	0	
customer_results	*merged	n/a	n/a	n/a	n/a	
samples	barcode_id	integer*	1000	6	0	
	order_id	integer*	1000	6	0	
	employee_id	integer*	1000	6	0	
	customer_id	integer*	1000	6	0	
	collection_date	date**	1000	6	0	
	collection_time	time**	1000	6	0	
containers	container_id	integer*	3	4	0	
	container_type	string	3	4	0	
	container_color	string	3	4	0	
	sample_type	string	3	4	0	
* changed dtypes from string representation of integers to actual integer type after discovery SQL autosequences for PK/FK						
** changed dtypes from string representation of dates/time to actual date/time type after discovery SQL now() function						

tables	columns	dtype	rows	cols	nulls	notes
analyzers	serial_id	string	4	3	0	
	make	string	4	3	0	
	model	string	4	3	0	
QC_panels	QC_panel_id	integer	27	4	0	Changed the names of the primary keys in all 3 'QC' tables to reflect the table names and restructure PK/FK.
	analyte_id	integer	27	4	0	
	serial_id	string	27	4	0	
	panel_id	integer	27	4	0	
QC_analytes	QC_analytes_id	integer	81	7	0	Changed the names of the primary keys in all 3 'QC' tables to reflect the table names and Added extra column as FK from QC_panels table after restructuring of PK/FK.
	manager_id	string	81	7	0	
	QC_panel_id	integer	81	7	0	
	QC_level	string	81	7	0	
	QC_range_low	numeric	81	7	0	
	QC_range_high	numeric	81	7	0	
QC_values	QC_mean	numeric	81	7	0	
	QC_values_id	integer*	567	5	0	
	QC_analytes_id	integer*	567	5	0	
	QC_value	numeric	567	5	0	
	QC_date	date**	567	5	0	
analytes	QC_time	time**	567	5	0	
	analytes_id	integer*	27	7	0	
	panel_id	integer*	27	7	0	
	panel_name	string	27	7	0	
	serial_id	string	27	7	0	
	analyte_name	string	27	7	0	
	analyte_mean	numeric	27	7	0	
	analyte_sd	numeric	27	7	0	
	units_of_measure	string	27	7	0	
panels	panel_id	integer*	7	4	0	
	container_id	integer*	7	4	0	
	panel_name	string	7	4	0	
	panel_charge	numeric	7	4	0	
charges	charge_id	integer*	1000	4	0	remove charge attribute from charges to create more normalization where it is inherited from panels so that if changes were needed they would be permuted via a FK
	panel_id	integer*	1000	4	0	
	customer_id	integer*	1000	4	0	
	order_id	integer*	1000	4	0	
	charge					

* changed dtypes from string representation of integers to actual integer type after discover autosequences for PK/FK in database design

** changed dtypes from string representation of dates/time to actual date/time type after discovery SQL now() function

tables	columns	dtype	rows	cols	nulls	notes
employee	employee_id	integer*	40	10	0	
	location_site	string	40	10	0	
	certification	string	40	10	30	
	email_id	integer*	40	10	0	
	phone_id	integer*	40	10	0	
	address_id	integer*	40	10	0	
	record_id	integer*	40	10	0	
	manager_id	string	40	10	0	
	first_name	string	40	10	0	
	last_name	string	40	10	0	
managers	manager_id	string	3***	1	0	see below
employment_record	record_id	integer*	40	3	0	
	department	string	40	3	0	
	salary	numeric	40	3	0	
address	address_id	integer*	545	7	0	Broke street into three components to be handled more efficiently by the web form in the html.
	street_number	string	545	7	0	
	street_name	string	545	7	0	
	street_suffix	string	545	7	0	
	city	string	545	7	0	
	state	string	545	7	0	
email_addresses	zip	string	545	7	0	
	email_id	integer*	540	2	0	
phone_number	email	string	540	2	0	
	phone_id	integer*	540	2	0	
	phone_number	string	540	2	0	

* changed dtypes from string representation of integers to actual integer type after discover autosequences for PK/FK in database design

*** added an "N/A" row to maintain the foreign key constraint with employee table since non-managers have n/a for manager_id in employees

Functional Requirements

(added to Section 1: Proposal to supplement or explain reasoning behind ER diagram formation)

An order of operation and cardinality exist between all the entities in the reference laboratory. Note, that this concept is key in developing the ER diagram that has been produced following this section. First, the cardinality is discussed between each entity one entity at a time:

Customers

Only one customer may have one address, email, and phone number listed.

Customers may place many orders but one order only consists of one customer.

A customer contributes many samples but only one sample pertains to one customer.

Customer information that must be recorded includes full name, date of birth, street, city state, zip code, email, and a phone number.

Orders

Orders are the centerpiece of the laboratory and initiated information being stored in many other tables. Specifically, when one and only customer submits an order through a form online, this triggers charges, customer tests, customer results, and samples to be collected. Specifically, each relationship is listed below:

An order is associated with only one charge (no variable pricing).

An order is associated with one 'test' panel and but many panels may be on one order

An order date and time must be initiated when a customer places an order.

Again, one order can only be associated with one customer but one customer may place many different orders. Note, one customer cannot place the same order at the same time though.

One order also prompts that one sample be collected for the order. One sample in this particular laboratory is never associated with many orders.

Orders also prompt customer tests to be triggered (through a laboratory information system) where many tests are associated with one order. In this laboratory, no order has just one test associated with it – customers must buy groups of tests called panels (cost effective).

Panels

Panels are ordered by customers and contain many analytes. Many different panels may be associated to one order, but one panel may never be ordered twice in one order.

Note that one panel consists of groups of analytes. For example, a renal panel consists of many analytes (BUN, CREATININE, GFR) used to measure kidney function. In this laboratory, one analyte may only be found on one panel (reduce redundant orders).

Each panel is associated with one particular container that is used to collect the one sample that is associated with one order. One container type though may be found on different panels. For example, a red top container type is used for the BMP and LIVER panels.

Finally, panels (groups of tests) are associated with quality control. Quality control is material run on analyzers with known analyte values to ensure integrity. There are three levels of quality control in this laboratory each associated with a panel. In other words, one panel may be found on many quality control types since many levels exist.

Note, panels must contain a panel name and a panel charge (amount charge for each panel).

Containers

A container is a device used to collect a particular panel. There are three different types of containers in this laboratory (red, blue, lavender). Each panel is only collected in one of the container types, however, one container type may be associated with one or more of the containers. Therefore the type of container needed depends on what panel is ordered.

For example, if a LIVER panel is ordered, then a red 'top' container is needed. If there are two different panels that each need the same type of container then two containers are drawn instead of one (ensures the customer has sufficient quantity for testing). For example, a LIVER and BMP are ordered so two red 'top' tubes will be drawn.

Quality Control

Again, quality control consists of three levels of material, each level for the one specific type of quality control test one panel. Since, One analyzer (device that measures analytes) may run many different types of quality control but again only one set of type of quality control per an analyzer.

Each panel also has three levels of quality control associated with it – low, medium, and high.

Each 'type' of quality control is specific to each panel test for each of the many analytes within that panel.

Therefore, each analyte has three levels of quality control associated with it – increasing concentration. Note, that by running quality control each individual analyte is verified per each day.

Quality control information needed to be recorded includes name of QC, the QC level, the value obtained, the date, and time QC was run. Note, that it is mandatory for laboratories to establish a mean value for each level of QC and that the laboratory must also create a low and high range that is usually (+/-) 2 SD from the mean where each analyte for each level of quality control would have its own 'range'.

Analyzers

Analyzers consist of analytes which are the main constituent of a customer test. A customer test consists of one analyte which results in one test result. For example, a potassium test has a result of 4.0 run on one specific analyzer in the laboratory.

Each analyzer runs many quality control types (one type per one panel) to ensure that the analytes contained on the panel, in this case a 'quality control panel' are within the quality control normal range.

Analyzers must have a unique serial id number, make, and model associated with them by law.

Analytes

Each analyte is associated with one customer test. An analyte is synonymous with the term reagent. Note, though analyte names also refer to the name of the substance that is being tested for in the customer's blood stream. A potassium reagent is used for a potassium test, to test for a potassium level in a customer.

Only one analyzer is used to test for any one specific analyte and subsequently the panel associated with the analyte. However, one analyzer may run many different types of panels consisting of many analytes. For example, a LIVER panel and all the analytes associated to the panel are run on the 'COBAS-C' analyzer', but the 'COBAS-C' may run other panels as well (i.e. BMP). However, these 'other' panels such as the BMP are not going to be found on other types of analyzers.

Similar to QC, each specific analyte must be tracked for a mean value as well as a standard deviation to establish a (+/-) 2SD patient range. The patient range denotes that 95% of the population would have a test result for that particular analyte between such a range. Analytes must also have a unit of measure associated by law, as well as a specific name where in this laboratory capitalized names represent the official analyte name on a test report as opposed to less formal version of an analyte name found in notes.

Customer Test

Each order consisting of one panel ordered by one customer triggers many tests for analytes contained on the specific panel.

The customer test becomes a test result. The test result is a value that is associated with a level of an analyte within the patient's blood. A test will never have many results.

A customer test is only performed by one employee (accountability). Obviously, one employee may perform many different tests though.

Customer Results

Customer results are the end product of a customer test. A customer result consists of a result value, date, and time. By law each test must contain some form of an alert to whether a value is considered high, normal, or low. This determination is almost always made by utilizing the 2SD range discussed in analytes.

Employees

Employees work at the laboratory and there exist three roles or types of employees.

First, the manager who manages employees and oversees customer ranges and quality control.

There are also techs who perform customer tests as mentioned above, and techs may or may not be deemed 'certified' – an attribute specific to techs. Note that by law customer tests must be associated with a tech (through an employee id) on test reports

Phlebotomists or collectors collect samples from customers. Note that lab collectors may work at various off site collection centers where this information is noted and lab collections must only be associated with collectors (through an employee id) on the sample

Employee information recorded by the company includes full name department (collections , analysis), street, city, state, zip, email, phone, and salary information is also recorded.

Samples

Samples are collected by collectors. A sample may only be associated with one collector however one collector may collect many samples.

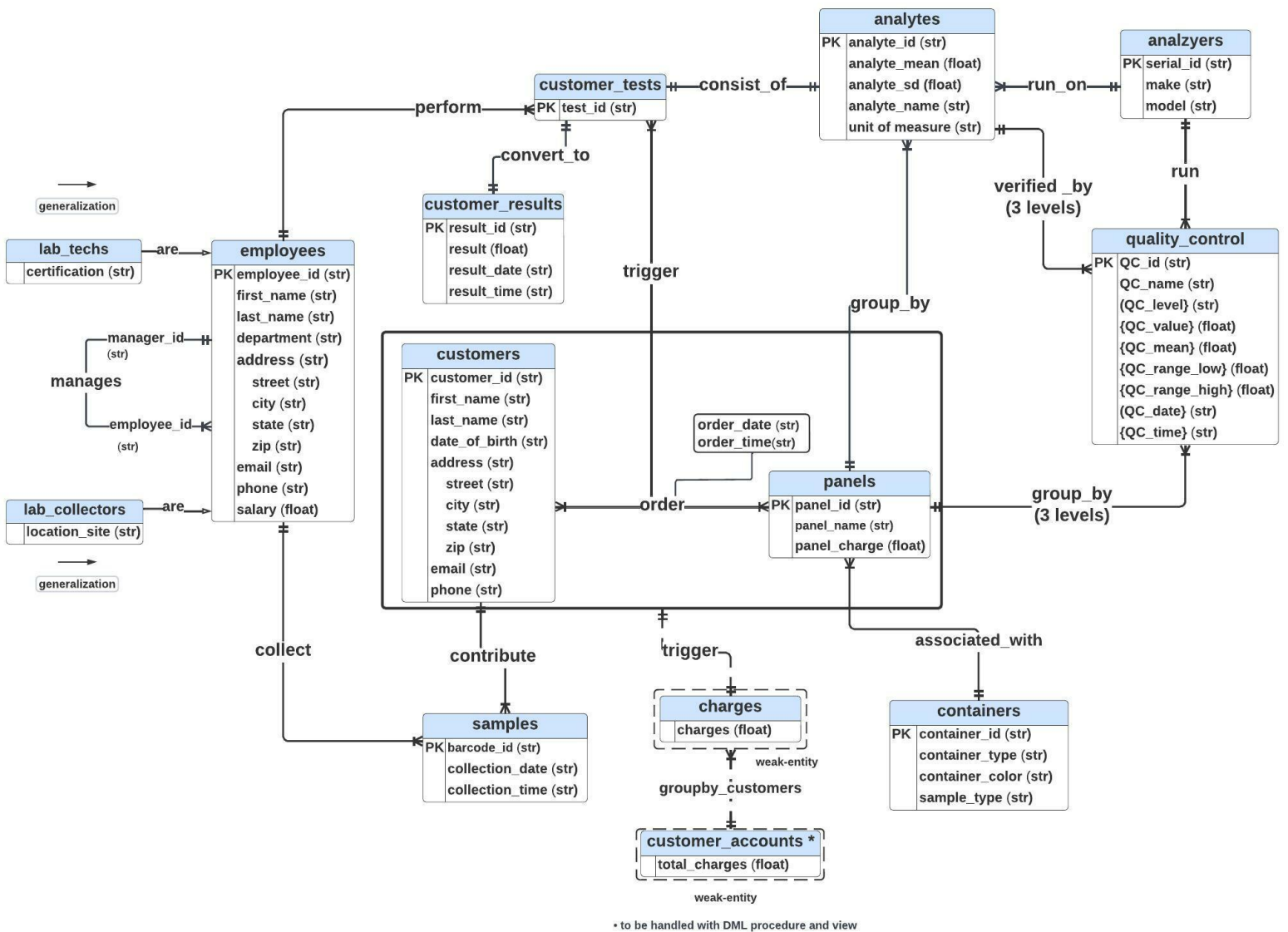
Again samples are triggered by orders. One order is associated with only one sample and one sample with one order.

Although samples are collected in containers, there exist many different types of containers that are dependent solely on what panel is ordered. Therefore containers are actually dependent on panels that have been ordered and not samples. A sample is just a patient's blood specimen and has no attributes to distinguish a specific type of container. Only order panels may do that.

Finally, as discussed in the beginning, an order of operation exists that pertains to one event triggering another . In this laboratory, when a customer orders a panel this triggers charges, samples to be drawn, and customer test to be run.

ER Diagram

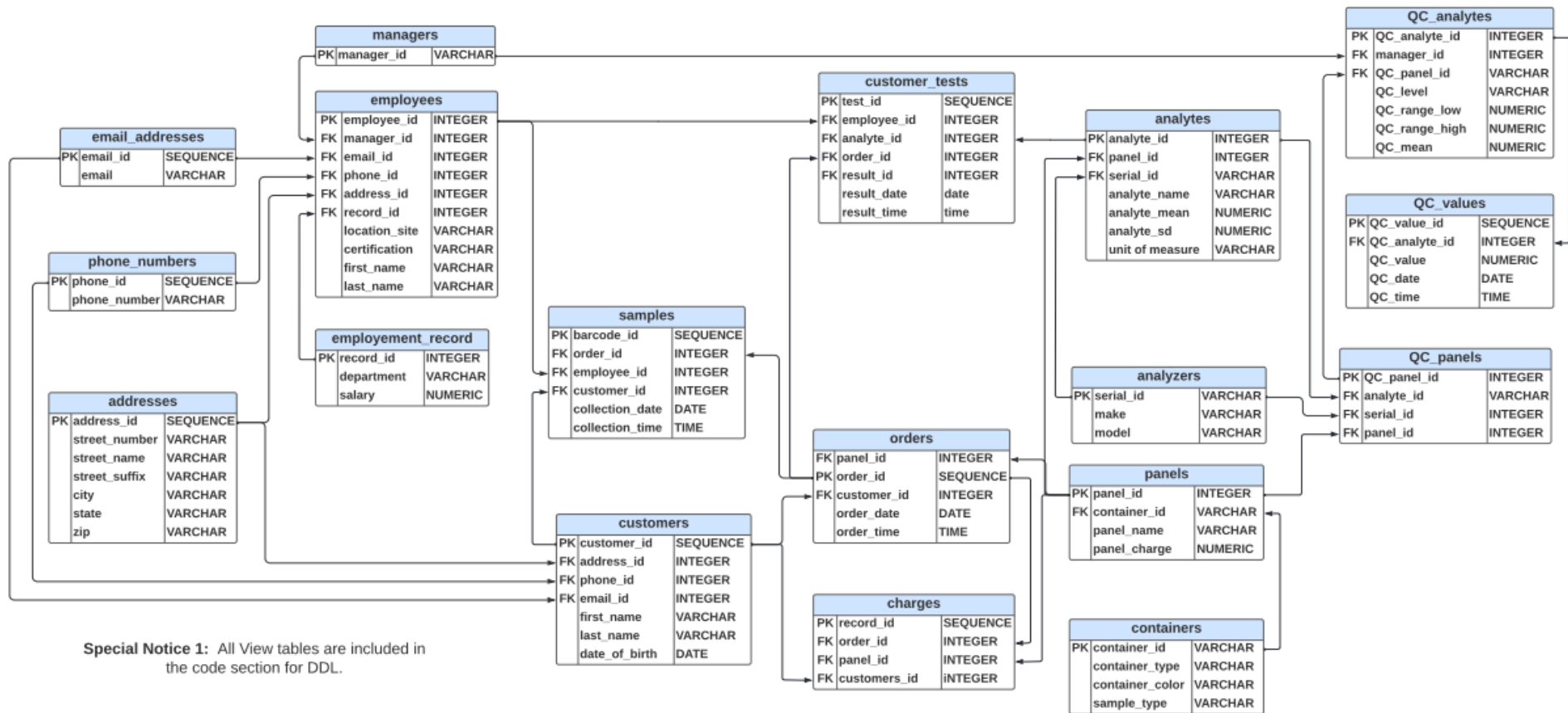
Conceptual Entity Relationship Diagram



Glossary for ER Diagram Tables :

- employees: Stores info specific to employees.
 - note: info from lab_techs and lab_collectors will be incorporated into employees in relational schema
- customer_tests: Stores info on what tests are ordered for customers.
- customer_results: Stores info on test results for customers.
- customers: Stores info on data related to customers.
- samples: Stores info relating specific to the samples used for testing.
- charges: tracks charges for individual orders (customer charged for a panel).
- customer_accounts: tracks aggregate sum of charges for each customer.
- panels: Stores information on the various panel names and associated costs – note a panel is a group of tests where customers purchase panels instead of individual tests. (i.e. customers will purchase a liver panel which will consist of different tests).
- containers: Stores information of the various types of tubes – this table is dependent on the panels table since panels require a specific type of container to collect samples.
- quality control: Stores information on all the names and results for many three different levels of quality control(QC). Labs test the accuracy of instruments by running fake QC samples with known values that run from low, medium, and high for every test in the lab. QC is organized into the same panel (groups of tests) sold to patients – basically testing the panel or 'product'. (This table will be broken up via normalization processes into separate entities in relational schema)
- analyzers: Stores information on the instruments used for testing samples.
- analytes: analytes are the individual tests associated with one panel. Note, each panel has a unique array of tests so no two panels are the same so customers can order many panels at one time.

Logical Schema Diagram



Although most tables are easy to discern from their names alone, three tables mentioned below are elaborated upon.:

QC_analytes serves as a dictionary to maintain the mean/sd for each qc analyte per each level of qc.

QC_panels serves a dictionary that maps QC panel names to each of the panels' individual analytes.

QC_values serves as a repository to hold all the values from the QC daily runs.

Software

1. Python (Jupyter) – raw data creation with NumPy, incorporating coding for SQL with Flask library, and any visualizations needed using Seaborn.
2. LucidChart⁵ – creation of Entity Relational and Relational Schema Diagrams
3. GitHub¹ – project repository used to share data with mentor and repository for uploads to Heroku servers
4. PostgreSQL using PGAdmin4⁶ – database DDL and DML language creation
5. Flask⁷ – web application design using Python
6. Heroku⁸ – web application hosting; alternatives: GitHub Pages⁹ ; Alternative 2: Hosting application on local machine and video taping - only in the worst case scenario with mentor approval.
7. Microsoft PowerPoint with Zoom to host presentation
 - Dependencies discussed more thoroughly in the discussion section.

Analysis plan

Analysis description

Detailed activities by Week has been developed and listed below:

Week	Activity Details
1	<ul style="list-style-type: none">• Create Topic for Project• Obtain All Necessary Software and Tools• Create Raw Data with Jupyter (Python)• Functional Requirements/ER Diagram/Relational Schema• Submit Proposal Due May 30th, 2022 23:59 EST
2	<ul style="list-style-type: none">• Perform Data Insertion of Raw Data• DDL Statements for Views and Entity Creation (SQL)
3	<ul style="list-style-type: none">• DML Statements for Queries and Modifications (SQL)• Test Database Formally with Various Query Statements
4-5	<ul style="list-style-type: none">• Integration Database with Web Application (Flask)• Web Application Hosting (Heroku)
6-7	<ul style="list-style-type: none">• Create PowerPoint Slides for Project• Project Presentation(Zoom)• Submit Project Materials Due July 3rd, 2022 23:59 EST

Weekly goals (personal deadlines noted in gray)

Week 1

- Create Topic for Project Personal Deadline: May 16th, 2022
- Obtain All Necessary Software and Tools Personal Deadline: May 18th, 2022
 - Download or create accounts for all software
 - Software list mentioned in Software section of this document)
- Create Raw Data with Jupyter (Python) Personal Deadline: May 16th, 2022
- Functional Requirements Personal Deadline: May 23th, 2022
- ER Diagram Personal Deadline: May 16th, 2022
- Relational Schema Personal Deadline: May 23rd, 2022
- Submit Proposal Personal Deadline: May 23rd, 2022 (Formal Deadline: May 30th, 2022)
 - Follow-up status of proposal by : May 27th , 2022
 - Make appropriate changes to Proposal: May 29th , 2022

Week 2 (Dependent on Proposal)

- Perform Data Insertion of Raw Data Personal Deadline: May 25th, 2022
- DDL Statements for Views and Entity Creation (SQL) Personal Deadline: May 28th, 2022
 - Entities from Relational Schema Personal Deadline: May 25th, 2022

Week 3

- DML Statements for Queries and Modifications (SQL) Personal Deadline: June 6th , 2022
Functions and procedures to be made for each Problem:
 - Problem 1: Personal Deadline: May 28th, 2022
 - Problem 2: Personal Deadline: May 29th, 2022
 - Problem 3: Personal Deadline: May 30th, 2022
 - Problem 4: Personal Deadline: May 31st, 2022
 - Problem 5: Personal Deadline: June 1st, 2022
 - Problem 6: Personal Deadline: June 2nd, 2022
 - Problem 7: Personal Deadline: June 3rd, 2022
 - Problem 8: Personal Deadline: June 4th, 2022
 - Problem 9: Personal Deadline: June 5th, 2022
 - Problem 10: Personal Deadline: June 6th, 2022
- Test Database Formally with Query Statements Personal Deadline: June 1st, 2022
 - Test and document ALL procedures, functions, and views with known cases of (null, 'negative', 'positive', 'incorrect data types', ect..) Personal Deadline: June 6th, 2022

Week 4 & 5

- Integration Database with Web Application (Flask)¹⁰ Personal Deadline: June 20th , 2022
 - Create a Directory and Files Personal Deadline: June 10th , 2022
 - Create 'Base' Environment Personal Deadline: June 12th , 2022
 - Add the HTML to web app (create 'templates') Personal Deadline: June 14th , 2022
 - Setup *Backend* SQL Personal Deadline: June 16th , 2022
 - 'Run' the Various Modules Personal Deadline: June 18th , 2022
 - Pray Everything Works – Serious Test Though! Personal Deadline: June 20th , 2022
- Web Application Hosting (Heroku)¹¹ Personal Deadline: June 25th , 2022
 - Compile All Code/Files on GitHub Repository Personal Deadline: June 20th , 2022
 - Create a Project within Heroku Personal Deadline: June 21st , 2022
 - Integrate Files to Heroku Server Personal Deadline: June 22nd , 2022
 - Deploy Application Personal Deadline: June 23rd , 2022
 - Verify Contents Personal Deadline: June 24th , 2022

Week 6 & 7

- Project Presentation(Zoom) Personal Deadline: June 28th , 2022
 - Create PowerPoint Slides for Project Personal Deadline: June 25th , 2022
 - Rehearse PowerPoint Slides for Project Personal Deadline: June 27th , 2022
- Submit Project Materials Personal Deadline: July 1st , 2022 (Formal Deadline: July 3rd , 2022)

Presentation plan

The database project is presented through Zoom video(recorded) using PowerPoint as a medium. The presentation is divided into three parts.

First, a brief background is given explaining the different parts of a reference laboratory basically as the different attributes to give the view a general knowledge about the topic.

Secondly, steps taken in the Analysis plan where key features of data creation, SQL database development, and web application hosting are explained. Various excerpts to the code, figures depicting data tables, and media such as figures are used to aid in the discussion.

Finally, the presentation will summarize by demonstrating the different web application functions that represent *solutions* to each of the ten problems listed in the Problem Objectives section. This also serves to document the successfully created solutions to each of the problems.

SECTION 2: POST-PROPOSAL FOLLOW-UP

The aim of section 2 is to provide a follow-up to the proposal and can be tied to the analysis plan that was initially placed in the proposal document. Section 2 will follow along with the document "Database Project Guidelines" to document success in providing the mandatory deliverables. The section is organized in a logical manner starting from creation of raw data, SQL statement creation, Flask source code functionality, and with the deployment of Heroku. More importantly, the role that each aforementioned modality above played in solving the problem set (10 enterprise needs) is assessed and discussed.

Raw Data Formation

Formation of raw data emulated the logical schema and was created using Pandas. Raw data was not crucially to solving the problem set specifically ; however, a quick mention of formation of data is mentioned below to provide a complete report of this project:

Formation of Columns:

Columns were simply made using both 'NumPy' and 'random' array functions that could create numerical or categorical data randomly or selectively. A table below summarizes these functions.

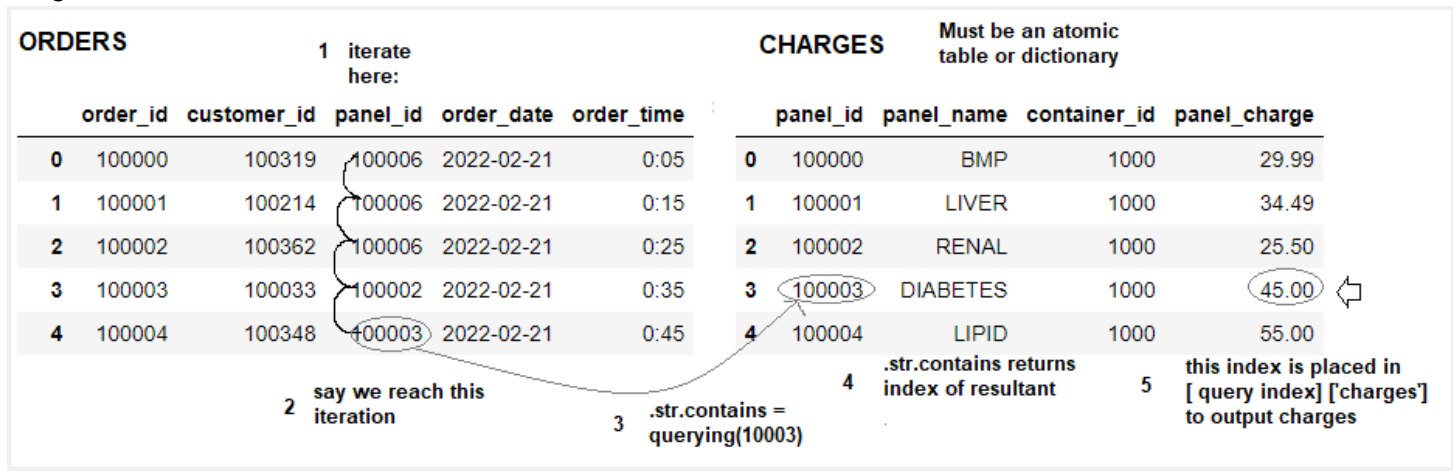
Data Needed	Function
categorical	random.choices
random integer	random.randint
sequence	[i for i in range()]
normal values	np.random.normal

Note, there were several columns that needed to correspond to a column in another table. This relationship was identical to a foreign-key constraint. For example, if a customer ordered a panel in the 'orders' table, these ordered panels must be translated to the appropriately corresponding charges. These charges are stored in a 'panels' table which serves as a dictionary relating one charge to one panel. The primary key in the 'panels' table is a foreign key in the orders table. Using this information we can map every instance of a panel_id in the orders table representing a customer chosen panel to the location of the row of the same panel_id in the panels table. Cleverly, if this location is then inserted into the 'panels' table as a 'mask' or 'index' we can decide to output the charge located on that same row. Every panel_id in the orders table then outputs a charge in the panels table and is so logical since charges and panel_id in panels table are atomic.

Below is the iteration with mapping used to produced the aforementioned effect:

```
for panel in orders['panel_id'].items():
    mapped_charges =
panels[panels['shared_col'].map(panel).str.contains(str(panel))]['charges'].tolist()
```

Diagram shows how function works:



Formation of Tables:

Tables were very simply made by inserting columns created above into a Pandas dataframe. These dataframes could be converted to .csv files by simply using .to_csv('name.csv') which then were used to create SQL insertion statements by a home-brewed Python's function mentioned below.

Conversion of Tables into SQL Insert Statements:

The following function simply takes a .csv file, string formats each row to appear like a SQL INSERT statement, and outputs each line into a text file. The .csv files were created using Pandas dataframes above using the .to_csv() function. Note, a text file was chosen over a .sql file to conserve the encoding as 'utf-8'.

def csv_file(csv, sql="DDL_INSERT.txt"):

```
import pandas as pd
df = pd.read_csv(csv + ".csv", header=None) # create df
file = open(sql, "a")
temp1 = [] # temp list to store row values
temp2 = [] # temp list to store entire formatted row
for cols in range(len(df)):
    for rows in df:
        if rows < (len(df.columns) - 1): # all instances except last column are formatted as such
            temp1.append(f'{df.iat[cols, rows]},')
        if rows == (len(df.columns) - 1): # when last column is reached, finish string formatting
            start = f"INSERT INTO {csv}\nVALUES("
            total = start + " ".join(temp1) + f'"{str(df.iat[cols, rows])}";\n'
            temp1.clear()
            temp2.append(total)
            del start
            del total
for t in temp2: # place formatted string = df row as one 'line' in the file
    print(t, file=file)
```

PostgreSQL Statements

(listed in order statement was enacted in database)

DDL Table Creation

Using the Logical Schema Diagram alongside , the database tables can be created to start to form a repository to hold the data created above.

Tables were all created using primary key constraints and relative data type seen in the Logical Schema. Foreign keys creation was performed separately as a last step in the DDL formation because constraint violations were persistent unless tables were created in an exact corresponding order to the dependencies on each table - so in short add foreign keys after. Although it is out of scope of the paper to list out every table creation, it is worth noting that on attributes that would never change or would need specific constraints - check constraints were given to these attributes. For example:

CHECK(serial_id in ('SN100000', 'SN100100','SN100300','SN100200')),

was used to maintain that only the latter depicted analyzer serial numbers can be entered. A full demonstration of all check constraints on attributes is seen in the Source Code Section at the end of the *Full Document* or alongside with provided files.

DML Inserts

The insertion statements for the project were created using the Python function mentioned in the Raw Data Section that produce a UTF-8 text file producing one such instance below:

```
INSERT INTO analyzers  
VALUES('SN100000', 'Roche', 'Cobas_C');
```

The function works smoothly and is nice because you can modify data very readily.

DML Sequences

Creating sequences and utilizing SERIAL data type in PostgreSQL was crucial to the functionality of the application used to accommodate the problem set. This is because SERIAL data type is able to automatically increment and sequences contain many psql functions that can be used to create clever functions within a trigger statement. This added the ability to create 'dynamic' primary keys that could autoincrement upon insertion of new data. This process is explained more in the *triggers* subsection below.

DDL Foreign Keys

Foreign keys are the lifeblood of the relational database. The interlink 2 or more tables in a corresponding fashion so that data can be kept normalized in individual tables, but also be interlinked in query statements that can be used to solve enterprise problems. In this project, many logical joins were performed between 2 or more tables on foreign key matches with primary keys in the parent table. The logical linkage provided most of the views created for the application that were used directly to solve the 10 problems listed in the Problems Objectives section. For example, linking orders, customers, charges, and panels and selecting necessary attributes allowed to create a customer account view that was logically created did not have to be created as one giant table in the

DML Inserts. This preserves data redundancy and facilitates *normalization* of the database by creating tables with rows only dependent on the primary key so when a mistake is made, a correction can be made and the problem does not permutate throughout instances that can not be linked.

DML Query (View) Tables

The DML Query tables are tables created from the integration of many tables in the database to provide answers to an enterprise's problems. These tables have columns that are necessary to perform key tasks in the enterprise such as creating a billing page or for my project providing a lab manager with a specific visualization needed to examine quality control (test if analyzers run properly).

The key component of formulating a *View Table* relies upon a properly formed schema relies upon the foreign key constraint and mapping between many tables that have primary key-foreign keys pairings. In this project, this mapping was primarily formed by left outer joins on the table with the primary key to create atomic corresponding joins with the child table. Atomicity is key in forming joins where the one table serves as a singular guide for the child table.

Moreover, statements such as aggregations can be incorporated as aliases to a corresponding atomic table tied to a foreign key; thus, providing even more solutions to an enterprise such as aggregate charges or getting the mean value for an analyte.

The *View Tables* can be elicited by Flask-SQLAlchemy to query specific attributes in the table via a web form where the results of the query may be displayed through a .html file on a web page. This is explained in more detail in the Flask-SQLAlchemy portion of this paper.

Listed below are the original problems of the reference laboratory with specific *View Table(s)* as a solution, and an brief explanation of how each view works:

Problem 1: charges needed to be grouped for each individual customer with aggregated sum

Solution 1: *customer_grouped_charges* and *customer_individual_charges*

Explanation: *Both an aggregated and non aggregated (individual) display of orders left joined on panels left joined on customers provide the perfect combination to create a billing view. Note the panels table is elicited because it maps one panel with a specific charge. Panels have a foreign key in orders and therefore can map charges to orders. Note, customers are also mapped via PK-FK pairing with orders.*

Problem 2: results need to be grouped by customer who should be able to filter by date and specific test

Solution 2: *customer_reports*

Explanation: *A left join on PK-FK pairing is also used here to join orders, with analyst, customers, and customer test providing all the columns needed for a laboratory report. Note, a report_GFR_table and reports_Alerts table are integrated with these tables. The first table is able to calculate a GFR value from a creatinine test from the customer's date of birth. The reports_Alerts table creates the joined mentioned originally but outputs a 'low', 'normal', and 'high' using a WHEN CASE statement that acts as a conditional with a 'return' statement.*

Problem 3: customers should be alerted to results that are deemed 'low' or 'high' with a relevant comment

Solution 3: *reports_alert_table* (note this table is merged into *customer_reports*)

Explanation: *As just mentioned the reports_Alerts table creates the joins mentioned originally but outputs a 'low', 'normal', and 'high' using a WHEN CASE statement that acts as a conditional with a 'return' statement. This function is incorporated into the customer reports functions as well .*

Problem 4: orders without results ('test to be run') or results with missing values should be made discoverable

Solution 4: *missing_tests_report*

Explanation: *A left outer join is performed on orders and customer tests where a WHEN CASE statement is used to conditionally check if a result exists in the customer test table with an order id in the same row and links this order id to the orders table so that a specific order can be tied to the check.*

Problem 5: calculated test reliant on patient demographics (i.e.age) should automatically calculate and insert

Solution 5: *reports_GFR_table*

Explanation: *Orders, customers, analytes, and tests are joined using PK-FK pairings to link all the information needed to elicit test results that are linked to the analyte 'CREATININE' as well to a specific customer's age*

(date_of_birth). All three are able to correspond because the orders table acts as an atomic parent table for all three where one single order_id number creates atomicity for all three tables. For example, we can say that this GFR is not tied to any other result (another order). The GFR value itself is calculated with help of the EXTRACT function.

Problem 6: lab collectors need to know what specific type of tubes are needed for each type of testing ordered
Solution 6: collector_containers

Explanation: The orders, containers, customers, samples, and customer tests tables are all left outer joined on PK-FK pairings. From this table a NOT IN statement is used to test if there is an order_id existing in the query (basically a customer has an order) but does not exist in the samples table (where collected sample data lives). The order id is input through a web form by the collector.

Problem 7: a barcode (QR code) should be created for each order per each customer

Solution 7: handled in flask simply using the orders and samples table

Explanation: This functionality is handled completely in the backend by Flask's object-oriented version of the reflected database because the function that produces the QR code is a Python library that works nicer with other Python functions.

Problem 8: QC results* that are out of range or violated specifically programmed rules must create alerts

Solution 8: manager_individual_qc_report and Pandas dataframe manipulations in Flask code

Explanation: This database contains three QC tables that each have a function to maintain normalization. Normalization is important in this scenario because QC means and analyte names are subject to change.

QC_analytes serves as a dictionary to maintain the mean/sd for each qc analyte per each level of qc.

QC_panels serves as a dictionary that maps QC panel names to each of the panels' individual analytes.

QC_values serves as a repository to hold all the values from the QC daily runs.

Linking all three tables together via logical joins provides the information needed to create a table that not only has a value but can have a column 'created' that houses the low and high range values.

Problem 9: QC results should populate by test or date unto a visualization called a Levy Jennings chart

Solution 9: manager_individual_qc_report and Python's Seaborn in Flask code.

Explanation: Here, Flask is used to integrate the manager_individual_qc_report with Seaborn scatterplot that uses the qc values plus to horizontal lines representing the (+/-) 2SD demarcations to form a graph that is colloquially known as a 'Levy-Jennings' plot. This same concept applies to financial graphs such as times series.

Problem 10: Cumulative statistics needed for each customer test to aid in creating reference ranges*

Solution 10: manager_range_reports

Explanation: An aggregation of all the customer test results by analyte can produce a mean and standard deviation value that provides a lab manager with a current aggregation value by analyte.

- Note in the Flask web application all tables above are integrated with web forms to allow users to choose specific attributes (i.e. analyte name) to work with each table.

Extra Problems solved:

An order form and subsequent trigger system mentioned below was created to make the web application more dynamic and this facilitates testing the functional solutions above much easiest.

DML Triggers

Any functional application needs to be able to respond to a change in the database. Classically, if a customer places an order, then inserts and updates may need to be provided in a charges table. For this project, the triggers served as a dynamic way to update the various *view tables* mentioned in the section above. The view tables serve as a way to accommodate the problems of an enterprise. With that said, the trigger is the means of maintaining these views in a dynamic or changing environment. The trigger functions created in this project were directly linked to the problem set and the query views or view tables listed below.

All the triggers first start when data is inserted into the orders table (from application through a web form). After insertion this prompts triggers to update the customer test, charges, and samples to create rows correspondingly through foreign key constraints. This mimics the actual flow of events in the reference laboratory. The newly inserted rows in samples, charges, and customer tests serve as a backbone to provide updates to the *view tables*.

Actually, triggers have been created for all the view tables when new data is inserted into charges, customer tests, or samples respectively. Since the view tables are data displayed in the web application, any time a customer places an order the web application is able to update what the customer or employee sees thus providing a dynamic solution to the problem set. The trigger functions enacted by the trigger definitions work to literally recreate the table views from the original queries.

Below, all 7 triggers are listed with a basic description of each usage (a complete list is provided as accompanied files and listed in the Source Code section in the *Full Document*):

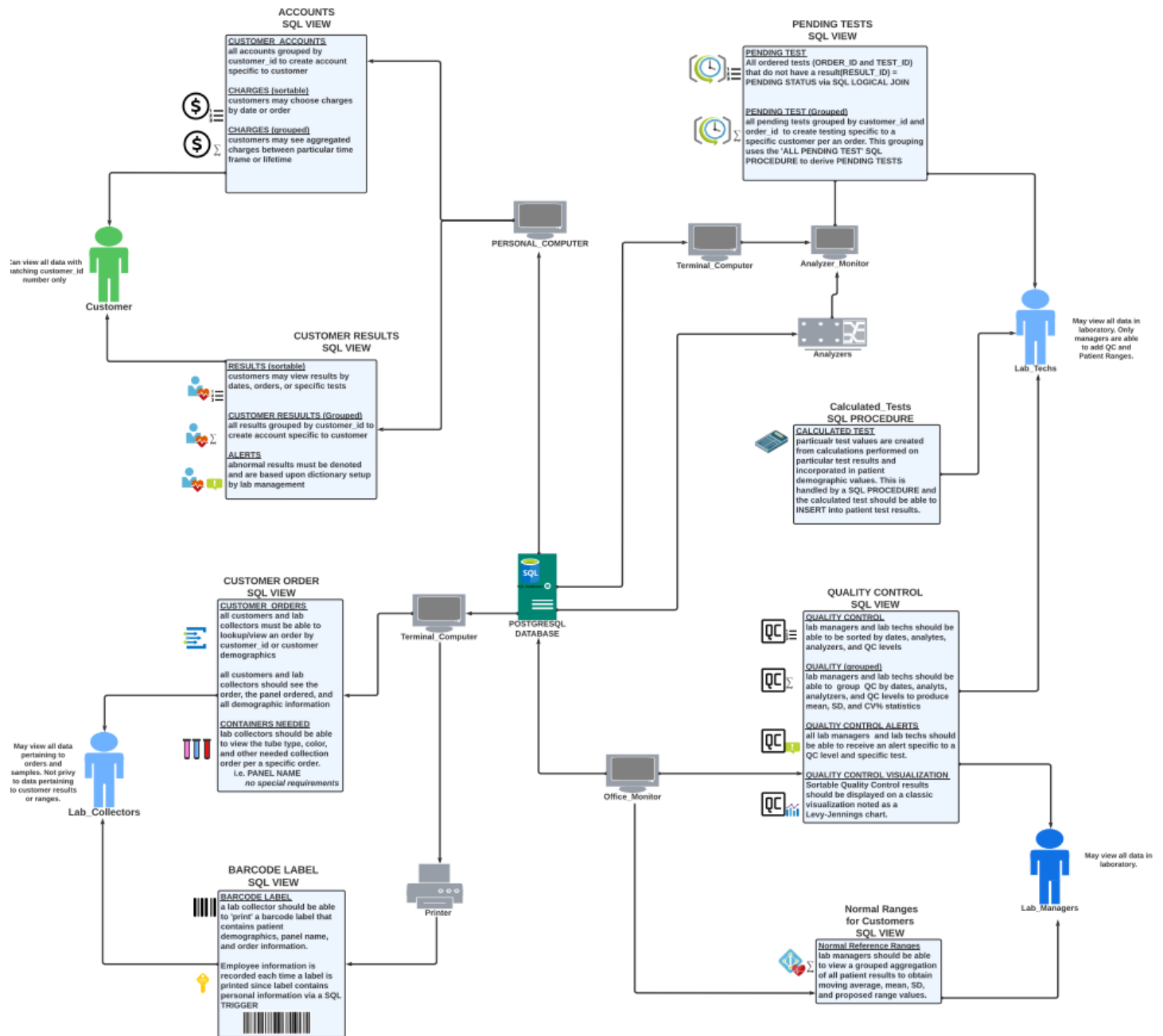
Name

collector_containers_trig_func()
customer_charges_trig_func()
customer_grouped_charges_trig_func()
customer_reports_trig()
orders_charges_trig_func()
orders_samples_trig_func()
orders_tests_trig_func()

Action

updates collector_containers view when samples is triggered
updates customer_charges view when charges is triggered
updates customer_grouped_charges view when charges is triggered
updates customer_reports_ view when customer testis triggered
updates orders_charges view when charges is triggered
updates orders_samples view when samples is triggered
updates orders_testsview when customer tests is triggered

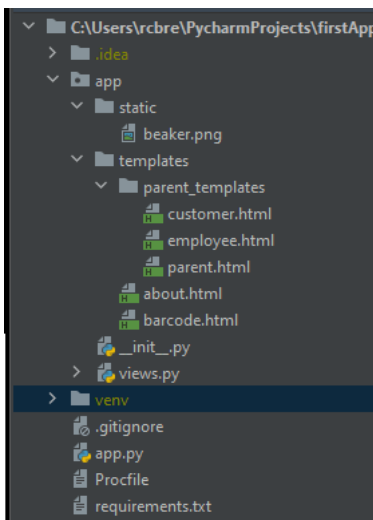
The infographic gives a depicted view of the SQL views discussed above:



Flask-SQLAlchemy - Application

In this section, the use of Flask-SQLAlchemy and more specific the functionalities of the library are discussed to further elaborate on how Python's Flask-SQLAlchemy may be used to further provide solutions to the enterprise problems suggested in the Problem Objectives and SQL View Table sections. Now, PostgreSQL statements provide the backbone or foundation to providing a system that can give the reference laboratory in this project views to work with. However, the database needs to be presented to the 'real-world' and Flask allows us to connect a database with all its views and constraints to a web server that uses the database in the backend while using .html files (or others) to display to the view.

Application Considerations



File Structure

When using Flask, file structure is paramount to a successful Heroku deployment as found out the hard way. The structure is needed for the Heroku/Flask to detect certain files at certain directory levels. Below, is the file structure in the Pycharm directory for this project

It is recommended that a user has a folder within the root directory where the root directory in my directory system is firstApp - the name of my app on my machine. Each of these file types are briefly described below with their respective role in promoting the services to the reference laboratory.

Dependencies

Tracking dependencies is key to ensuring that in the future that code can function. Classically, a pip freeze > requirements.txt is used in Windows to create a file with all dependencies after pip installing each.

Virtualenv

Python's library virtualenv (naturally installed on Pycharm) allows each project including this one to maintain its own specific dependencies. This folder should be included in the .gitignore folder though -Heroku uses requirements.txt to get dependencies.

Git Repository

Most projects don't exist without being tracked within a Git repository. The hidden folder keeps a meta-dictionary tracking/auditing all changes to a file. The repository serves as the 'file source' that is sent to Heroku so it is mandatory.

Procfile

a fileless file (yes) that houses the information for the name of the server to use and the name of the application.

Flask Object

After discussing vital items needed to successfully host the application of these users may be able to use the View we return to discussing the actual specific problem set and how Flask promotes user abstraction to allow enteripes tasks to be completed with ease.

One of the major if not most important classes in Flask is the Flask class. The object actually plays an important role in this project and for organization because it allows all information in the database to be used within the realms of DDL/DML statements. More specifically, Flask converts this database into an object-oriented relational model (OORM) where the user may now use Python to manipulate data any way that suits an enterprise's needs. On top of this it also allows all the data to then be hosted on a server.

Instantiating the class so that 'app' will integrate with many other methods.

```
app = Flask(__name__)
```

Database Reflection and Automap

Two options exist when using Flask, one may build the database schema with the Flask objects or *reflect or automap* from an already existing database. I chose the latter and found this to be an extremely useful time saver in the project.

Specific to this project, this allowed me to convert all database queries into dataframes that in turn could be turned into .html format with the Pandas `to_html()` method. This provided way more capabilities of data wrangling for this project where I could use all the Panda data wrangling functions alongside any other Python library. Personally, I prefer Python over SQL because it is object-oriented as opposed to procedural.

app.routes(),

A 'route' is the actual view on the server and links together the .html file to be hosted on a designated URL and does this through the app decorated (`@app.route`) that utilizes are the class methods of the Flask() class to integrate this process.

```
@app.route('/someURL')
def some_function():
    return render_template('someHTML.html', variable=ViewTableAfterPandasWrangling)
```

render_template(),

As seen above, `render_template` promotes the View Tables to be displayed to a URL serving as a guided user interface (GUI) where the level of abstraction is taken away letting employees and customers focus on the task they need to complete in the real world. Web forms seem to play a huge role in allowing users to enter data into the database while html templates allow users to view queries - specifically the *View Tables* created in PostgreSQL.

Jinja templates

Employees and customers in the reference laboratory use client server model where we ask "GET" or receive "POST" and both of these demands have variables that can tie in the data between the server side and the database all with Flask_SQLAlchemy as the keystone

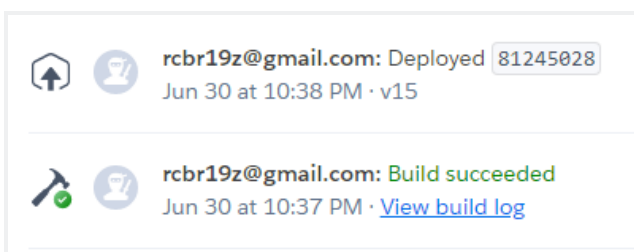
centerpiece that uses object-oriented programming to relay server console commands with the database - perhaps, you could even call it off shoot of a *middleware*.

Heroku

After all the work is done to provide the reference laboratory with a level of abstraction that can facilitate all the demands requested by this enterprise, the application itself must be hosted.

Simply, push your source code to the Heroku remote repository after logging in (command prompt automates this):

```
Login to Heroku
$ heroku login
Get to project directory
$ cd my-project/
Initialize a git repository in a new or existing directory
$ git init
Add remote repository
$ heroku git:remote -a mock-laboratory
Add files to repository (use '.' for adding all files)
$ git add .
Commit project (if no more changes)
$ git commit -am "make it better"
Deploy master branch to Heroku.
$ git push heroku master
```



After problem solving that an extra space in the Procfile causes first failure, the website specifically hosts. Listed on the next several pages are all the views located on the website. This website can be access and used (dynamically) at the URL:

<https://mock-laboratory.herokuapp.com/>

Project Links

Website

<https://mock-laboratory.herokuapp.com/>

GitHub page

<https://github.com/RyansStacks/Capstone-Database-Project-DTSC691->

Website Screenshots

In this section I make mention of the fact that a video has been created that orchestrates that the trigger functions and website work by recording database tables and web views before and order. After, the order is placed and all relevant tables populate with appropriate data.

Specifically, when an order is placed the orders, samples, customer tests, and charges tables are appropriately populated as well as the view tables that the latter mentioned tables trigger. For this reason we see an appropriate change on the web site as well. This video has been saved in my **Google Drive (this video is called 'Breen_Project_TriggerProofs')**. Note, this video is separate from my 'Walkthrough' video where I simply show the user all the functionalities of the web site at the end. Please feel free to try this proof as well at the website :

<https://mock-laboratory.herokuapp.com/>.

Cited References

The following documents that particular sections of code within the files (in bold) noted below were adopted from outside sources and should be given credit for. Note, all code notes mentioned were only referenced and created with complete origination.

sql files

5_DML_Triggers_UTF8_B (orders_charges_trig_func()) function

"postgresql - How to insert stored functions and triggers in a Postgres container arrow_drop_down"

<https://stackoverflow.com/questions/67658462/how-to-insert-stored-functions-and-triggers-in-a-postgres-container>

5_DML_Triggers_UTF8_B (orders_charges) table

What to return from a PostgreSQL row level trigger? - CYBERTEC

<https://www.cybertec-postgresql.com/en/what-to-return-from-a-postgresql-row-level-trigger/>

html files

employee.html

stylesheet for html obtained from bootstrap

<https://getbootstrap.com/docs/4.2/getting-started/download/>

parent.html

stylesheet for html obtained from bootstrap

<https://getbootstrap.com/docs/4.2/getting-started/download/>

customer.html

stylesheet for html obtained from bootstrap

<https://getbootstrap.com/docs/4.2/getting-started/download/>

https://github.com/CoreyMSchafer/code_snippets/blob/master/Django_Blog/03Templates/django_project/blog/templates/blog/base.html

<https://stackoverflow.com/questions/41513463/bootstrap-align-navbar-items-to-the-right>

Python

Pandas Docs

<https://pandas.pydata.org/docs/>

qr_code

<https://pypi.org/project/qrcode/>

Seaborn

<https://seaborn.pydata.org/>

Flask SQLAlchemy

Automapping Setup:

<https://docs.sqlalchemy.org/en/14/orm/extensions/automap.html>

Automap Tutorial

<https://www.youtube.com/watch?v=UK57IHhSh8I>

Project Source Coude

SQL Statements

1_DDL_TABLE_UTF8

```
CREATE TABLE analyzers(  
  serial_id      varchar  
    NOT NULL  
    CHECK (serial_id in ('SN100000','SN100100','SN100300','SN100200')),  
  make          varchar  
    NOT NULL CHECK(make <>'')  
    CHECK (make in ('Roche', 'ACL', 'Sysmex')),  
  model         varchar  
    NOT NULL CHECK(model <>'')  
    CHECK (model in ('Cobas_C', 'Cobas_I', 'Tops', 'H1000')),  
  CONSTRAINT analyzers_pkey PRIMARY KEY(serial_id)  
);
```

```
CREATE TABLE QC_panels(  
  QC_panel_id   integer  
    NOT NULL  
    CHECK (QC_panel_id BETWEEN 100000 AND 100027),  
  analyte_id    integer  
    NOT NULL  
    CHECK (analyte_id BETWEEN 100000 AND 100027),  
  serial_id     varchar  
    NOT NULL CHECK(serial_id <>'')  
  CHECK(serial_id in ('SN100000', 'SN100100','SN100300','SN100200')),  
  panel_id      integer  
    NOT NULL  
    CHECK (panel_id BETWEEN 100000 AND 100006),  
  CONSTRAINT QC_panels_pkey PRIMARY KEY(QC_panel_id)  
);
```

```
CREATE TABLE QC_analytes(  
  QC_analyte_id integer  
    NOT NULL  
    CHECK (QC_analyte_id BETWEEN 100000 AND 100081),  
  QC_panel_id   integer  
    NOT NULL  
    CHECK (QC_panel_id BETWEEN 100000 AND 100027),  
  QC_level      varchar  
    NOT NULL CHECK(QC_level <>'')  
    CHECK (QC_level in ('1','2','3')),  
  manager_id    varchar  
    NOT NULL CHECK(manager_id <>'')  
    CHECK (CAST(manager_id as int) BETWEEN 100000 AND 999999),  
  QC_range_low  numeric  
    NOT NULL CHECK(CAST(QC_range_low as varchar) <> '')  
    CHECK ( QC_range_low > 0),  
  QC_mean       numeric  
    NOT NULL CHECK(CAST(QC_mean as varchar) <> '')  
    CHECK ( QC_mean > 0) ,  
  QC_range_high numeric  
    NOT NULL CHECK(CAST(QC_range_high as varchar) <> '')  
    CHECK ( QC_range_high > 0),  
  CONSTRAINT QC_analytes_pkey PRIMARY KEY(QC_analyte_id)  
);
```

```
CREATE TABLE QC_values(  
  QC_value_id   integer  
    NOT NULL  
    CHECK (QC_value_id BETWEEN 100000 AND 999999),  
  QC_analyte_id integer  
    NOT NULL  
    CHECK (QC_analyte_id BETWEEN 100000 AND 100081),  
  QC_value      numeric  
    NOT NULL CHECK(CAST(QC_value as varchar) <>'')  
    CHECK (QC_value > 0),  
  QC_date       date  
    NOT NULL CHECK(CAST(QC_date as varchar) <>''),  
  QC_time       time  
    NOT NULL CHECK(CAST(QC_time as varchar) <>''),  
  CONSTRAINT QC_values_pkey PRIMARY KEY(QC_value_id)  
);
```

```
CREATE TABLE containers(  
  container_id   integer  
    NOT NULL  
    CHECK (container_id BETWEEN 1000 AND 1040),  
  container_type varchar  
    CHECK (container_type in ('tube')),  
  container_color varchar  
    CHECK (container_color in ('blue','red','lavendar')),  
  sample_type    varchar  
    CHECK (sample_type in ('plasma', 'serum', 'whole blood')),  
  CONSTRAINT containers_pkey PRIMARY KEY(container_id)  
);
```

```
CREATE TABLE panels(  
  panel_id      integer  
    NOT NULL  
    CHECK (panel_id BETWEEN 100000 AND 100006),  
  CONSTRAINT panels_pkey PRIMARY KEY(panel_id)  
);
```

```

panel_id      integer
NOT NULL
CHECK (panel_id BETWEEN 100000 AND 100007),
panel_name    varchar
CHECK (panel_name in ('BMP', 'LIVER', 'RENAL', 'DIABETES', 'LIPID', 'CBC', 'COAG')),
container_id  integer
NOT NULL
CHECK (container_id BETWEEN 1000 AND 1040),
panel_charge  numeric
NOT NULL CHECK(CAST(panel_charge as varchar) <>'') CHECK ( panel_charge > 0),
CONSTRAINT panels_pkey PRIMARY KEY(panel_id)
);

CREATE TABLE analytes(
analyte_id    integer
NOT NULL
CHECK (analyte_id BETWEEN 100000 AND 100027),
serial_id     varchar
NOT NULL CHECK(serial_id <>'')
CHECK (serial_id in ('SN100000', 'SN100100', 'SN100300', 'SN100200')),
panel_id      integer
NOT NULL
CHECK (panel_id BETWEEN 100000 AND 999999),
analyte_name  varchar
NOT NULL CHECK(analyte_name <>''),
analyte_mean  numeric
NOT NULL CHECK(CAST(analyte_mean as varchar) <>'') ,
analyte_sd    numeric
NOT NULL CHECK(CAST(analyte_sd as varchar)<>''),
units_of_measure varchar
NOT NULL CHECK(units_of_measure <>'')
CHECK (units_of_measure in ('mg/dL', 'sec', 'g/dL', '%', 'U/L', '/mL', 'mmol/L', 'g/L', 'mL/min/1.73 m²')),
CONSTRAINT analytes_pkey PRIMARY KEY(analyte_id)
);

CREATE TABLE orders(
order_id      serial,
customer_id   integer,
panel_id      integer,
order_date    date,
order_time    time,
CONSTRAINT orders_pkey PRIMARY KEY(order_id)
);

CREATE TABLE charges(
charge_id     serial
NOT NULL
CHECK (charge_id BETWEEN 100000 AND 999999),
panel_id      integer
NOT NULL
CHECK (panel_id BETWEEN 100000 AND 999999),
order_id      serial
NOT NULL
CHECK (order_id BETWEEN 100000 AND 999999),
customer_id   integer
NOT NULL
CHECK (order_id BETWEEN 100000 AND 999999)
);

CREATE TABLE customer_tests(
test_id       serial
NOT NULL
CHECK (test_id BETWEEN 100000 AND 999999),
employee_id   integer
NOT NULL
CHECK (employee_id BETWEEN 100000 AND 999999),
analyte_id    integer
NOT NULL
CHECK (analyte_id BETWEEN 100000 AND 999999),
order_id      integer
NOT NULL
CHECK (order_id BETWEEN 100000 AND 999999),
result        numeric
NOT NULL CHECK(CAST(result as varchar) <>''),
result_date    date
NOT NULL CHECK(CAST(result_date as varchar) <>''),
result_time    time
NOT NULL CHECK(CAST(result_time as varchar) <>''),
CONSTRAINT customer_tests_pkey PRIMARY KEY(test_id)
);

CREATE TABLE customers(
customer_id   serial,
address_id    serial,
phone_id      serial,
email_id      serial,
first_name    varchar,
last_name     varchar ,
date_of_birth date,
CONSTRAINT customers_pkey PRIMARY KEY(customer_id)
);

CREATE TABLE samples(
barcode_id    serial
NOT NULL
CHECK (barcode_id BETWEEN 100000 AND 999999),
order_id      integer

```

```

NOT NULL
CHECK (order_id BETWEEN 100000 AND 999999),
employee_id      integer
NOT NULL
CHECK (employee_id BETWEEN 100000 AND 999999),
customer_id      integer
NOT NULL
CHECK (customer_id BETWEEN 100000 AND 999999),
collection_date   date
NOT NULL CHECK(CAST(collection_date as varchar) <>''),
collection_time    time
NOT NULL CHECK(CAST(collection_time as varchar) <>''),
CONSTRAINT samples_pkey PRIMARY KEY(barcode_id)
);

CREATE TABLE employees(
employee_id      integer
NOT NULL
CHECK (employee_id BETWEEN 100000 AND 999999),
location_site    varchar
NOT NULL CHECK(location_site <>'')
CHECK (location_site in ('100541', '100542', '100543', '100544', '100545')),
certification     varchar ,
email_id          integer
NOT NULL
CHECK (email_id BETWEEN 100000 AND 999999),
phone_id          integer
NOT NULL
CHECK (phone_id BETWEEN 100000 AND 999999),
address_id        integer
NOT NULL
CHECK (address_id BETWEEN 100000 AND 999999),
record_id         integer
NOT NULL
CHECK (record_id BETWEEN 100000 AND 999999),
manager_id        varchar
NOT NULL CHECK(manager_id <>'')
CHECK(manager_id in ('Not Applicable', '100000', '100001')),
first_name        varchar
NOT NULL CHECK(first_name <>'')
CHECK (first_name ~ '^[A-Z].*$'),
last_name         varchar
NOT NULL CHECK(last_name <>'')
CHECK (last_name ~ '^[A-Z].*$'),
CONSTRAINT employees_pkey PRIMARY KEY(employee_id)
);

CREATE TABLE managers(
manager_id        varchar
NOT NULL CHECK(manager_id <>'')
CHECK (manager_id in ('100000', '100001', 'Not Applicable' )),
CONSTRAINT managers_pkey PRIMARY KEY(manager_id)
);

CREATE TABLE employment_records(
record_id         integer
NOT NULL
CHECK (record_id BETWEEN 100000 AND 999999),
department        varchar
CHECK (department in ('specimen collection', 'laboratory')),
salary            numeric
NOT NULL CHECK(CAST(salary as varchar) <>'')
CHECK (salary > 20000),
CONSTRAINT employment_record_pkey PRIMARY KEY(record_id)
);

CREATE TABLE addresses(
address_id        serial NOT NULL
CHECK (address_id BETWEEN 100000 AND 999999),
street_number     varchar NOT NULL ,
street_name       varchar NOT NULL ,
street_suffix     varchar NOT NULL ,
city              varchar NOT NULL ,
state             varchar NOT NULL ,
zip               varchar NOT NULL ,
CONSTRAINT address_pkey PRIMARY KEY(address_id)
);

CREATE TABLE email_addresses(
email_id          serial NOT NULL
CHECK (email_id BETWEEN 100000 AND 999999),
email             varchar NOT NULL ,
CONSTRAINT email_pkey PRIMARY KEY(email_id)
);

CREATE TABLE phone_numbers(
phone_id          serial NOT NULL
CHECK (phone_id BETWEEN 100000 AND 999999),
phone_number      varchar,
CONSTRAINT phone_number_pkey PRIMARY KEY(phone_id)
);

```

3_DML_SEQUENCES_UTF8

```
CREATE SEQUENCE orders_sequence
START 101000
INCREMENT 1;
ALTER TABLE orders
ALTER COLUMN order_id
SET DEFAULT nextval('orders_sequence');

CREATE SEQUENCE QC_values_sequence
START 100567
INCREMENT 1;
ALTER TABLE QC_values
ALTER COLUMN QC_value_id
SET DEFAULT nextval('QC_values_sequence');

CREATE SEQUENCE charges_sequence
START 101000
INCREMENT 1;
ALTER TABLE charges
ALTER COLUMN charge_id
SET DEFAULT nextval('charges_sequence');

CREATE SEQUENCE customer_tests_sequence
START 103847
INCREMENT 1;
ALTER TABLE customer_tests
ALTER COLUMN test_id
SET DEFAULT nextval('customer_tests_sequence');

CREATE SEQUENCE customers_sequence
START 100500
INCREMENT 1;
ALTER TABLE customers
ALTER COLUMN customer_id
SET DEFAULT nextval('customers_sequence');

CREATE SEQUENCE customers_address_sequence
START 100545
INCREMENT 1;
ALTER TABLE customers
ALTER COLUMN address_id
SET DEFAULT nextval('customers_address_sequence');

CREATE SEQUENCE customers_email_sequence
START 100540
INCREMENT 1;
ALTER TABLE customers
ALTER COLUMN email_id
SET DEFAULT nextval('customers_email_sequence');

CREATE SEQUENCE customers_phone_sequence
START 100540
INCREMENT 1;
ALTER TABLE customers
ALTER COLUMN phone_id
SET DEFAULT nextval('customers_phone_sequence');

CREATE SEQUENCE samples_sequence
START 101000
INCREMENT 1;
ALTER TABLE samples
ALTER COLUMN barcode_id
SET DEFAULT nextval('samples_sequence');

CREATE SEQUENCE addresses_sequence
START 100545
INCREMENT 1;
ALTER TABLE addresses
ALTER COLUMN address_id
SET DEFAULT nextval('addresses_sequence');

CREATE SEQUENCE email_addresses_sequence
START 100540
INCREMENT 1;
ALTER TABLE email_addresses
ALTER COLUMN email_id
SET DEFAULT nextval('email_addresses_sequence');

CREATE SEQUENCE phone_numbers_sequence
START 100540
INCREMENT 1;
ALTER TABLE phone_numbers
ALTER COLUMN phone_id
SET DEFAULT nextval('phone_numbers_sequence');
```

4_DDL_FOREIGNKEYS_UTF8

```
/* Foreign Keys for Laboratory Database */
```

```
/* QC_panels FK */
ALTER TABLE QC_panels
```

```

ADD CONSTRAINT QC_panels_analyzers_fkey
FOREIGN KEY (serial_id)
REFERENCES analyzers (serial_id)
ON DELETE CASCADE;

ALTER TABLE QC_panels
ADD CONSTRAINT QC_panels_analytes_fkey
FOREIGN KEY (analyte_id)
REFERENCES analytes (analyte_id)
ON DELETE CASCADE;

ALTER TABLE QC_panels
ADD CONSTRAINT QC_panels_panels_fkey
FOREIGN KEY (panel_id)
REFERENCES panels(panel_id)
ON DELETE CASCADE;

/* QC_analytes FK */
ALTER TABLE QC_analytes
ADD CONSTRAINT QC_analytes_managers_fkey
FOREIGN KEY (manager_id)
REFERENCES managers(manager_id)
ON DELETE CASCADE;

ALTER TABLE QC_analytes
ADD CONSTRAINT QC_analytes_panels_fkey
FOREIGN KEY (QC_panel_id)
REFERENCES QC_panels(QC_panel_id)
ON DELETE CASCADE;

/* QC_values FK */
ALTER TABLE QC_values
ADD CONSTRAINT QC_values_QC_analytes_fkey
FOREIGN KEY (QC_analyte_id)
REFERENCES QC_analytes(QC_analyte_id)
ON DELETE CASCADE;

/* analytes FK */
ALTER TABLE analytes
ADD CONSTRAINT analytes_panels_fkey
FOREIGN KEY (panel_id)
REFERENCES panels(panel_id)
ON DELETE CASCADE;

ALTER TABLE analytes
ADD CONSTRAINT analytes_analyzers_fkey
FOREIGN KEY (serial_id)
REFERENCES analyzers(serial_id)
ON DELETE CASCADE;

/* panels FK */
ALTER TABLE panels
ADD CONSTRAINT panels_containers_fkey
FOREIGN KEY (container_id)
REFERENCES containers(container_id)
ON DELETE CASCADE;

/* charges FK */
ALTER TABLE charges
ADD CONSTRAINT charges_panels_fkey
FOREIGN KEY (panel_id)
REFERENCES panels(panel_id)
ON DELETE CASCADE;

ALTER TABLE charges
ADD CONSTRAINT charges_orders_fkey
FOREIGN KEY (order_id)
REFERENCES orders(order_id)
ON DELETE CASCADE;

ALTER TABLE charges
ADD CONSTRAINT charges_customers_fkey
FOREIGN KEY (customer_id)
REFERENCES customers(customer_id)
ON DELETE CASCADE;

/* customers FK */
ALTER TABLE customers
ADD CONSTRAINT customers_addresses_fkey
FOREIGN KEY (address_id)
REFERENCES addresses(address_id)
ON DELETE CASCADE;

ALTER TABLE customers
ADD CONSTRAINT customers_phone_numbers_fkey
FOREIGN KEY (phone_id)
REFERENCES phone_numbers(phone_id)
ON DELETE CASCADE;

ALTER TABLE customers
ADD CONSTRAINT customers_email_addresses_fkey
FOREIGN KEY (email_id)
REFERENCES email_addresses(email_id)
ON DELETE CASCADE;

/* orders FK */
ALTER TABLE orders
ADD CONSTRAINT orders_panels_fkey
FOREIGN KEY (panel_id)

```

```

REFERENCES panels(panel_id)
ON DELETE CASCADE;

ALTER TABLE orders
ADD CONSTRAINT orders_customers_fkey
FOREIGN KEY (customer_id)
REFERENCES customers(customer_id)
ON DELETE CASCADE;

/* customer_tests FK */
ALTER TABLE customer_tests
ADD CONSTRAINT customer_tests_employee_fkey
FOREIGN KEY (employee_id)
REFERENCES employees(employee_id)
ON DELETE CASCADE;

ALTER TABLE customer_tests
ADD CONSTRAINT customer_tests_analytes_fkey
FOREIGN KEY (analyte_id)
REFERENCES analytes(analyte_id)
ON DELETE CASCADE;

ALTER TABLE customer_tests
ADD CONSTRAINT customer_tests_orders_fkey
FOREIGN KEY (order_id)
REFERENCES orders(order_id)
ON DELETE CASCADE;

/* samples FK */
ALTER TABLE samples
ADD CONSTRAINT samples_orders_fkey
FOREIGN KEY (order_id)
REFERENCES orders(order_id)
ON DELETE CASCADE;

ALTER TABLE samples
ADD CONSTRAINT samples_employees_fkey
FOREIGN KEY (employee_id)
REFERENCES employees(employee_id)
ON DELETE CASCADE;

ALTER TABLE samples
ADD CONSTRAINT samples_customers_fkey
FOREIGN KEY (customer_id)
REFERENCES customers(customer_id)
ON DELETE CASCADE;

/* employees FK */
ALTER TABLE employees
ADD CONSTRAINT employees_managers_fkey
FOREIGN KEY (manager_id)
REFERENCES managers(manager_id)
ON DELETE CASCADE;

ALTER TABLE employees
ADD CONSTRAINT employees_email_addresses_fkey
FOREIGN KEY (email_id)
REFERENCES email_addresses(email_id)
ON DELETE CASCADE;

ALTER TABLE employees
ADD CONSTRAINT employees_phone_numbers_fkey
FOREIGN KEY (phone_id)
REFERENCES phone_numbers(phone_id)
ON DELETE CASCADE;

ALTER TABLE employees
ADD CONSTRAINT employees_addresses_fkey
FOREIGN KEY (address_id)
REFERENCES addresses(address_id)
ON DELETE CASCADE;

ALTER TABLE employees
ADD CONSTRAINT employees_employment_records_fkey
FOREIGN KEY (record_id)
REFERENCES employment_records(record_id)
ON DELETE CASCADE;

```

5_DML_Triggers_UTF8_B

```

/* updates charges after */
CREATE OR REPLACE FUNCTION orders_charges_trig_func()
RETURNS TRIGGER
LANGUAGE PLPGSQL
AS
$$
BEGIN
INSERT INTO charges
SELECT nextval('charges_sequence') as charge_id,
new.panel_id,
new.order_id,
new.customer_id
FROM orders, charges_sequence
ORDER BY new.order_id desc
LIMIT 1;
RETURN NULL;
END;

```



```

$$;

CREATE TRIGGER orders_charges
AFTER INSERT ON orders
FOR EACH ROW
EXECUTE PROCEDURE orders_charges_trig_func();

/* updates customer tests after insert into orders*/
CREATE OR REPLACE FUNCTION orders_tests_trig_func()
RETURNS TRIGGER
LANGUAGE PLPGSQL
AS
$$
BEGIN
INSERT INTO customer_tests
SELECT nextval('customer_tests_sequence') as test_id,
o.employee_id,
a.analyte_id,
o.order_id,
ROUND(a.analyte_mean::numeric + (random()/2)::numeric, 1) AS result,
o.result_date,
o.result_time
FROM(SELECT
floor(random()*(100035-100000+1))+100000 as employee_id,
orders.order_id,
orders.panel_id,
orders.order_date as result_date,
orders.order_time as result_time
FROM orders, customer_tests_sequence
ORDER BY orders.order_id desc
LIMIT 1) as o
LEFT OUTER JOIN
(SELECT analytes.panel_id,
analytes.analyte_id,
analytes.analyte_mean
FROM panels
LEFT OUTER JOIN analytes
ON panels.panel_id = analytes.panel_id) as a
ON o.panel_id = a.panel_id;
RETURN NULL;
END;
$$;
CREATE TRIGGER orders_tests
AFTER INSERT ON orders
FOR EACH ROW
EXECUTE PROCEDURE orders_tests_trig_func();

/* updates samples table after insert into orders */
CREATE OR REPLACE FUNCTION orders_samples_trig_func()
RETURNS TRIGGER
LANGUAGE PLPGSQL
AS
$$
BEGIN
INSERT INTO samples
SELECT nextval('samples_sequence') as barcode_id,
NEW.order_id,
floor(random()*(100040-100000+1))+100000 as employee_id,
NEW.customer_id,
NEW.order_date as collection_date,
orders.order_time as collection_time
FROM orders, samples_sequence
ORDER BY NEW.order_id desc
LIMIT 1;
RETURN NULL;
END;
$$;
CREATE TRIGGER orders_samples
AFTER INSERT ON orders
FOR EACH ROW
EXECUTE PROCEDURE orders_samples_trig_func();

/* Collector Container Trigger */
CREATE OR REPLACE FUNCTION collector_containers_trig_func()
RETURNS TRIGGER
LANGUAGE PLPGSQL
AS
$$
BEGIN

DROP TABLE collector_containers ;
DROP MATERIALIZED VIEW collector_containers_view;

CREATE MATERIALIZED VIEW collector_containers_view
AS
SELECT row_number() OVER (ORDER BY samples.barcode_id,orders.customer_id) AS unique_id,
orders.order_id, orders.customer_id, customers.first_name, customers.last_name,
panels.panel_name, samples.barcode_id, containers.container_type, containers.container_color,
containers.sample_type
FROM orders
LEFT OUTER JOIN panels
ON orders.panel_id = panels.panel_id
LEFT OUTER JOIN containers
ON panels.container_id = containers.container_id
LEFT OUTER JOIN customers
ON orders.customer_id = customers.customer_id
LEFT OUTER JOIN samples
ON samples.order_id = orders.order_id

```

```

LEFT OUTER JOIN customer_tests
ON orders.order_id = customer_tests.order_id
WHERE orders.order_id
NOT IN
(SELECT samples.order_id
FROM samples);

CREATE TABLE collector_containers
AS
SELECT *
FROM collector_containers_view
ORDER BY unique_id;
ALTER TABLE ONLY collector_containers
ADD CONSTRAINT collector_containers_pk PRIMARY KEY(unique_id );

RETURN NULL;
END;
$$;

CREATE TRIGGER collector_containers_trig
AFTER INSERT ON samples
FOR EACH ROW
EXECUTE PROCEDURE collector_containers_trig_func();

/* Customer Charges Trigger */
CREATE OR REPLACE FUNCTION customer_charges_trig_func()
RETURNS TRIGGER
LANGUAGE PLPGSQL
AS
$$
BEGIN

DROP TABLE customers_individual_charges;
DROP MATERIALIZED VIEW customers_charges_discrete;

CREATE MATERIALIZED VIEW customers_charges_discrete
AS
SELECT row_number() OVER (ORDER BY orders.order_id,orders.customer_id) AS unique_id,
orders.customer_id, orders.order_id, customers.first_name, customers.last_name, panels.panel_charge
FROM orders
LEFT OUTER JOIN panels
ON orders.panel_id = panels.panel_id
LEFT OUTER JOIN customers
ON orders.customer_id = customers.customer_id;

CREATE TABLE customers_individual_charges
AS
SELECT *
FROM customers_charges_discrete
ORDER BY unique_id;
ALTER TABLE ONLY customers_individual_charges
ADD CONSTRAINT customers_individual_charges_pk PRIMARY KEY(unique_id );

RETURN NULL;
END;
$$;

CREATE TRIGGER customer_charges_trig
AFTER INSERT ON charges
FOR EACH ROW
EXECUTE PROCEDURE customer_charges_trig_func();

/* Customer Grouped Charges Trigger */
CREATE OR REPLACE FUNCTION customer_grouped_charges_trig()
RETURNS TRIGGER
LANGUAGE PLPGSQL
AS
$$
BEGIN

DROP TABLE customer_grouped_charges;
DROP MATERIALIZED VIEW customer_charges_grouped;

CREATE MATERIALIZED VIEW customer_charges_grouped
AS
SELECT orders.customer_id, customers.first_name, customers.last_name, SUM(panels.panel_charge)
FROM orders
LEFT OUTER JOIN panels
ON orders.panel_id = panels.panel_id
LEFT OUTER JOIN customers
ON orders.customer_id = customers.customer_id
GROUP BY orders.customer_id, customers.first_name, customers.last_name;

CREATE TABLE customer_grouped_charges
AS
SELECT *
FROM customer_charges_grouped
ORDER BY customer_id;
ALTER TABLE ONLY customer_grouped_charges
ADD CONSTRAINT customer_grouped_charges_pk PRIMARY KEY(customer_id );

RETURN NULL;
END;
$$;

CREATE TRIGGER customer_charges_grouped_trig
AFTER INSERT ON charges
FOR EACH ROW
EXECUTE PROCEDURE customer_grouped_charges_trig();

/* Customer Reports Trigger */
CREATE OR REPLACE FUNCTION customer_reports_trig()

```

```

    RETURNS TRIGGER
    LANGUAGE PLPGSQL
    AS
$$
BEGIN

DROP MATERIALIZED VIEW IF EXISTS report_GFR;
DROP TABLE IF EXISTS report_GFR_table;
DROP MATERIALIZED VIEW IF EXISTS report_alerts;
DROP TABLE IF EXISTS report_alerts_table;
DROP TABLE IF EXISTS customer_reports;

/* Calculate GFR */
CREATE MATERIALIZED VIEW report_GFR
AS
SELECT orders.order_id, analytes.analyte_id,
ROUND(EXTRACT(YEARS FROM AGE(NOW(),customers.date_of_birth))/10 +
customer_tests.result*10 + 60,1) as GFR
FROM customers
RIGHT OUTER JOIN orders
ON customers.customer_id = orders.customer_id
LEFT OUTER JOIN customer_tests
ON customer_tests.order_id = orders.order_id
LEFT OUTER JOIN analytes
ON analytes.analyte_id = customer_tests.analyte_id
WHERE analytes.analyte_name = 'CREATININE';

CREATE TABLE report_GFR_table
AS
SELECT * FROM report_GFR ;

/* create table with alerts relative to customer results (+/- 2SD)*/
CREATE MATERIALIZED VIEW report_alerts
AS
SELECT row_number() OVER (ORDER BY orders.order_id,orders.customer_id) AS unique_id,
orders.order_id, orders.customer_id, customers.first_name,
customers.last_name, analytes.analyte_name, customer_tests.result,
analytes.units_of_measure, customer_tests.result_date, customer_tests.result_time,
CASE
WHEN (customer_tests.result - analytes.analyte_mean) > (2*analytes.analyte_sd)
THEN 'High'
WHEN (customer_tests.result - analytes.analyte_mean) < (-2*analytes.analyte_sd)
THEN 'Low'
ELSE 'Normal'
END alert
FROM orders
LEFT OUTER JOIN customer_tests
ON customer_tests.order_id = orders.order_id
LEFT OUTER JOIN customers
ON customers.customer_id = orders.customer_id
LEFT OUTER JOIN analytes
ON customer_tests.analyte_id = analytes.analyte_id;

CREATE TABLE report_alerts_table
AS SELECT * FROM report_alerts;
/* merge two tables into one final customer report */
UPDATE report_alerts_table
SET result = report_GFR_table.gfr
FROM report_GFR_table
WHERE report_alerts_table.analyte_name = 'GFR'
AND report_GFR_table.order_id = report_alerts_table.order_id;

CREATE TABLE customer_reports
AS
SELECT *
FROM report_alerts_table
ORDER BY unique_id;

ALTER TABLE ONLY customer_reports
ADD CONSTRAINT customer_reports_pk PRIMARY KEY(unique_id );

RETURN NULL;
END;
$$;

CREATE TRIGGER customer_reports_trig
AFTER INSERT ON customer_tests
FOR EACH ROW
EXECUTE PROCEDURE customer_reports_trig();

```

6_DML_Query_Table

```

/* Collector Views */
/* Containers Needed */
CREATE MATERIALIZED VIEW collector_containers_view
AS
SELECT row_number() OVER (ORDER BY samples.barcode_id,orders.customer_id) AS unique_id,
orders.order_id, orders.customer_id, customers.first_name, customers.last_name,
panels.panel_name, samples.barcode_id, containers.container_type, containers.container_color,
containers.sample_type
FROM orders

```

```

LEFT OUTER JOIN panels
ON orders.panel_id = panels.panel_id
LEFT OUTER JOIN containers
ON panels.container_id = containers.container_id
LEFT OUTER JOIN customers
ON orders.customer_id = customers.customer_id
LEFT OUTER JOIN samples
ON samples.order_id = orders.order_id
LEFT OUTER JOIN customer_tests
ON orders.order_id = customer_tests.order_id
WHERE orders.order_id
NOT IN
(SELECT samples.order_id
FROM samples);

CREATE TABLE collector_containers
AS
SELECT *
FROM collector_containers_view
ORDER BY unique_id;
ALTER TABLE ONLY collector_containers
ADD CONSTRAINT collector_containers_pk PRIMARY KEY(unique_id );

/* Customer Views */

/* Customer Individual Charges */
CREATE MATERIALIZED VIEW customers_charges_discrete
AS
SELECT row_number() OVER (ORDER BY orders.order_id,orders.customer_id) AS unique_id,
orders.customer_id, orders.order_id, customers.first_name, customers.last_name, panels.panel_charge
FROM orders
LEFT OUTER JOIN panels
ON orders.panel_id = panels.panel_id
LEFT OUTER JOIN customers
ON orders.customer_id = customers.customer_id;

CREATE TABLE customers_individual_charges
AS
SELECT *
FROM customers_charges_discrete
ORDER BY unique_id;
ALTER TABLE ONLY customers_individual_charges
ADD CONSTRAINT customers_individual_charges_pk PRIMARY KEY(unique_id );

/* Customer Grouped Charges */

CREATE MATERIALIZED VIEW customer_charges_grouped
AS
SELECT orders.customer_id, customers.first_name, customers.last_name, SUM(panels.panel_charge)
FROM orders
LEFT OUTER JOIN panels
ON orders.panel_id = panels.panel_id
LEFT OUTER JOIN customers
ON orders.customer_id = customers.customer_id
GROUP BY orders.customer_id, customers.first_name, customers.last_name;

CREATE TABLE customer_grouped_charges
AS
SELECT *
FROM customer_charges_grouped
ORDER BY customer_id;
ALTER TABLE ONLY customer_grouped_charges
ADD CONSTRAINT customer_grouped_charges_pk PRIMARY KEY(customer_id );

/* Customer Reports */

/* Calculate GFR */
DROP MATERIALIZED VIEW IF EXISTS report_GFR;
CREATE MATERIALIZED VIEW report_GFR
AS
SELECT orders.order_id, analytes.analyte_id,
ROUND(EXTRACT(YEARS FROM AGE(NOW(),customers.date_of_birth))/10 +
customer_tests.result*10 + 60,1) as GFR
FROM customers
RIGHT OUTER JOIN orders
ON customers.customer_id = orders.customer_id
LEFT OUTER JOIN customer_tests
ON customer_tests.order_id = orders.order_id
LEFT OUTER JOIN analytes
ON analytes.analyte_id = customer_tests.analyte_id
WHERE analytes.analyte_name = 'CREATININE';

DROP TABLE IF EXISTS report_GFR_table;
CREATE TABLE report_GFR_table
AS
SELECT * FROM report_GFR ;

/* create table with alerts relative to customer results (+/- 2SD)*/
DROP MATERIALIZED VIEW IF EXISTS report_alerts;
CREATE MATERIALIZED VIEW report_alerts
AS
SELECT row_number() OVER (ORDER BY orders.order_id,orders.customer_id) AS unique_id,
orders.order_id, orders.customer_id, customers.first_name,
customers.last_name, analytes.analyte_name, customer_tests.result,
analytes.units_of_measure, customer_tests.result_date, customer_tests.result_time,
CASE
WHEN (customer_tests.result - analytes.analyte_mean) > (2*analytes.analyte_sd)
THEN 'High'
WHEN (customer_tests.result - analytes.analyte_mean) < (-2*analytes.analyte_sd)
THEN 'Low'
ELSE 'Normal'

```

```

END alert
FROM orders
LEFT OUTER JOIN customer_tests
ON customer_tests.order_id = orders.order_id
LEFT OUTER JOIN customers
ON customers.customer_id = orders.customer_id
LEFT OUTER JOIN analytes
ON customer_tests.analyte_id = analytes.analyte_id;

DROP TABLE IF EXISTS report_alerts_table;
CREATE TABLE report_alerts_table
AS SELECT * FROM report_alerts;

/* merge two tables into one final customer report */
UPDATE report_alerts_table
SET result = report_GFR_table.gfr
FROM report_GFR_table
WHERE report_alerts_table.analyte_name = 'GFR'
AND report_GFR_table.order_id = report_alerts_table.order_id;

DROP TABLE IF EXISTS customer_reports;
CREATE TABLE customer_reports
AS
SELECT *
FROM report_alerts_table
ORDER BY unique_id;

ALTER TABLE ONLY customer_reports
ADD CONSTRAINT customer_reports_pk PRIMARY KEY(unique_id );

/* Missing Tests */
CREATE MATERIALIZED VIEW missing_tests
AS
SELECT orders.order_id,
       customers.customer_id,
       analytes.analyte_name,
       customer_tests.result,
       CASE
         WHEN analytes.analyte_name IS NULL AND customer_tests.result IS NULL
         THEN 'Order Did Not Crossover'
         WHEN analytes.analyte_name IS NOT NULL AND customer_tests.result IS NULL
         THEN 'Test Has Not Yet Resulted'
         ELSE 'Programming Error'
       END explanation
FROM customers
RIGHT OUTER JOIN orders
ON customers.customer_id = orders.customer_id
LEFT OUTER JOIN customer_tests
ON customer_tests.order_id = orders.order_id
LEFT OUTER JOIN analytes
ON analytes.analyte_id = customer_tests.analyte_id
WHERE customer_tests.result is NULL;

CREATE TABLE missing_tests_report
AS
SELECT *
FROM missing_tests
ORDER BY order_id;
ALTER TABLE ONLY missing_tests_report
ADD CONSTRAINT missing_tests_report_pk PRIMARY KEY(order_id);

/* Manager Views */
/* Manager Individual Q.C. */
CREATE MATERIALIZED VIEW manager_individual_QC AS
SELECT row_number() OVER (ORDER BY QC_analytes.QC_level, analytes.analyte_name) AS unique_id,
       QC_values.QC_date,   QC_values.QC_time,   QC_analytes.QC_level,
       analytes.analyte_name, QC_values.QC_value, analytes.units_of_measure,
       analyzers.make,    analyzers.model,
       CASE
         WHEN (QC_values.QC_value > QC_analytes.QC_range_high)
         THEN 'High'
         WHEN (QC_values.QC_value < QC_analytes.QC_range_low)
         THEN 'Low'
         ELSE 'Normal'
       END alert
FROM QC_analytes
LEFT OUTER JOIN QC_values
ON QC_analytes.QC_analyte_id = QC_values.QC_analyte_id
LEFT OUTER JOIN QC_panels
ON QC_analytes.QC_panel_id = QC_panels.QC_panel_id
LEFT OUTER JOIN analytes
ON QC_panels.analyte_id = analytes.analyte_id
LEFT OUTER JOIN analyzers
ON QC_panels.serial_id = analyzers.serial_id;

CREATE TABLE manager_individual_QC_report
AS
SELECT *
FROM manager_individual_QC
ORDER BY unique_id;
ALTER TABLE ONLY manager_individual_QC_report
ADD CONSTRAINT manager_individual_QC_report_pk PRIMARY KEY(unique_id );

/* Manager Grouped Q.C. */
CREATE MATERIALIZED VIEW manager_grouped_QC AS
SELECT row_number() OVER (ORDER BY qc_level, analyte_name) AS unique_id,

```

```

        qcp.qc_level,
        analytes.analyte_name,
        qcp.mean,
        qcp.sd,
        analytes.units_of_measure,
        analyzers.make,
        analyzers.model
FROM analytes, analyzers,
(SELECT *
FROM QC_panels,
(SELECT *
FROM QC_analytes,
(SELECT QC_values.QC_analyte_id,
        ROUND(AVG(QC_values.QC_value),1) as Mean,
        ROUND(STDDEV_SAMP(QC_values.QC_value),1) as SD
FROM QC_values
RIGHT OUTER JOIN QC_analytes
ON QC_analytes.QC_analyte_id = QC_values.QC_analyte_id
GROUP BY QC_values.QC_analyte_id) AS agg
WHERE QC_analytes.qc_analyte_id = agg.qc_analyte_id) as qca
WHERE qca.qc_panel_id = QC_panels.qc_panel_id) as qcp
WHERE qcp.analyte_id = analytes.analyte_id
      AND qcp.serial_id = analyzers.serial_id
ORDER BY qcp.qc_level;

CREATE TABLE manager_grouped_QC_report
AS
SELECT *
FROM manager_grouped_QC
ORDER BY unique_id;
ALTER TABLE ONLY manager_grouped_QC_report
ADD CONSTRAINT  manager_grouped_QC_report_pk PRIMARY KEY(unique_id );


/* Manager Patient Ranges */
CREATE MATERIALIZED VIEW ranges AS
SELECT  agg.analyte_name,
        agg.count as N,
        agg.mean_actual,
        agg.sd_actual,
        analytes.analyte_mean as mean_target,
        analytes.analyte_sd as sd_target,
        analyzers.make,
        analyzers.model
FROM analytes,analyzers,
(SELECT analytes.analyte_name,
        COUNT(customer_tests.analyte_id),
        ROUND(AVG(customer_tests.result),2) AS mean_actual,
        ROUND(STDDEV_SAMP(customer_tests.result),2) as sd_actual,
        analytes.units_of_measure
FROM orders
LEFT OUTER JOIN customer_tests
ON customer_tests.order_id = orders.order_id
LEFT OUTER JOIN analytes
ON customer_tests.analyte_id = analytes.analyte_id
GROUP BY analytes.analyte_name, analytes.units_of_measure) as agg
WHERE agg.analyte_name = analytes.analyte_name
AND analytes.serial_id = analyzers.serial_id;

CREATE TABLE manager_range_report
AS
SELECT *
FROM ranges
ORDER BY analyte_name;

ALTER TABLE ONLY manager_range_report
ADD CONSTRAINT  manager_range_report_pk PRIMARY KEY(analyte_name);

```

APPLICATION CODE

FILE STRUCTURE FOR NAMED FILES BELOW STARTING AT ROOT DIRECTORY:

'-' indicates one subdirectory level

indentation underneath folder is a subfolder

```

Root_Folder
  firstApp(folder)
  -app(folder)
    --static(folder)
    --template(folder)
      --- parent_templates
        ---- parent html files(employee, customer, parent)
        --- all other html files live within the template folder
    -- __init__.py
    -- views.py

  requirements.txt
  Procfile
  app.py

```

in firstApp folder (lives inside the root directory - in my case is was Pycharm

app.py

```
In [ ]:  from app import app

        # run with debugger off for deploy
        if __name__ == '__main__':
            app.run(debug=False)
```

Procfile

```
In [ ]:  web: gunicorn app:app
```

requirements.txt

```
In [ ]:  click==8.1.3
        colorama==0.4.5
        cyclер==0.11.0
        DateTime==4.4
        Flask==2.1.2
        Flask-SQLAlchemy==2.5.1
        fonttools==4.33.3
        greenlet==1.1.2
        gunicorn==20.1.0
        itsdangerous==2.1.2
        Jinja2==3.1.2
        kiwisolver==1.4.3
        MarkupSafe==2.1.1
        matplotlib==3.5.2
        numpy==1.23.0
        packaging==21.3
        pandas==1.4.3
        Pillow==9.1.1
        pycopg2==2.9.3
        pyparsing==3.0.9
        python-dateutil==2.8.2
        pytz==2022.1
        qrcode==7.3.1
        scipy==1.8.1
        seaborn==0.11.2
        six==1.16.0
        SQLAlchemy==1.4.39
        Werkzeug==2.1.2
        zope.interface==5.4.0
```

.gitignore

```
In [ ]:  venv
        .idea
        app/__pycache__/
```

in app folder (lives inside firstAPP folder) Note: firstApp lives inside directory

__init__.py

```
In [ ]:  from flask import Flask

        # Initiate app
        app = Flask(__name__)

        from app import views
```

views.py

```

In [ ]: ❸ from flask import Flask, render_template, request
import pandas as pd
import numpy as np
import scipy.stats
import qrcode
import os
import io
import base64
import seaborn as sns
import matplotlib
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy.ext.automap import automap_base

matplotlib.use('Agg')
import matplotlib.pyplot as plt
from datetime import datetime

# import app from init.py
from app import app

# correct for Heroku mistake of no ql in URI
DATABASE_URI = os.environ['DATABASE_URL']
DATABASE_URI = DATABASE_URI[:8]+'ql' + DATABASE_URI[8:]
app.config['SQLALCHEMY_DATABASE_URI'] = DATABASE_URI
app.config['SQL_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)
# Map schema
Base = automap_base()
Base.prepare(db.engine, reflect=True)

# HOME PAGE
@app.route('/')
def index():
    return render_template("index.html")

# ABOUT THIS PROJECT
@app.route('/about')
def about():
    return render_template("about.html")

# ORDERS
@app.route('/orders')
def orders_home():
    return render_template('orders.html')

@app.route('/place_order', methods=['POST', 'GET'])
def place_orders():
    if request.method == 'GET':
        return 'You must submit the form to access this age'
    if request.method == 'POST':
        complete = request.form
        complete = len(complete)
        if complete < 12:
            return render_template('incorrect.html', display="You have not fully completed the form!")
        else:
            dictionary = request.form
            panels = [k for k, v in dictionary.items() if v == 'on']
            first_name = request.form.get('first_name')
            first_name = first_name.title()
            last_name = request.form.get('last_name')
            last_name = last_name.title()
            date_of_birth = request.form.get('date_of_birth')
            street_number = request.form.get('street_number')
            street_name = request.form.get('street_name').title()
            street_name = street_name.title()
            street_suffix = request.form.get('street_suffix').title()
            city = request.form.get('city')
            city = city.title()
            state = request.form.get('state')
            state = state.title()
            zip_code = request.form.get('zip')
            email = request.form.get('email')
            phone = request.form.get('phone')
            # Is customer in DB?

            Customers1 = Base.classes.customers
            customers1 = db.session.query(Customers1).filter_by(first_name=first_name, last_name=last_name,
                                                                date_of_birth=date_of_birth).count()

            Addresses1 = Base.classes.addresses
            addresses1 = db.session.query(Addresses1).filter_by(street_number=street_number, street_name=street_name,
                                                                street_suffix=street_suffix, city=city, state=state,
                                                                zip=zip_code).count()

            Phone_Numbers1 = Base.classes.phone_numbers
            phone_numbers1 = db.session.query(Phone_Numbers1).filter_by(phone_number=phone).count()

            Email_Addresses1 = Base.classes.email_addresses
            email_addresses1 = db.session.query(Email_Addresses1).filter_by(email=email).count()
            # existing customers (in database under following tables):
            if customers1 > 0 and addresses1 > 0 and phone_numbers1 > 0 and email_addresses1 > 0:
                Customers1 = Base.classes.customers
                customers1 = db.session.query(Customers1.customer_id).filter_by(first_name=first_name,
                                                                                last_name=last_name,
                                                                                date_of_birth=date_of_birth).first()

                customer_id = customers1[0]
                current = datetime.now()
                order_date = current.strftime('%Y-%m-%d')

```



```

        order_time = current.strftime('%H:%M:%S')
        Orders1 = Base.classes.orders
        last = db.session.query(Orders1).order_by(Orders1.order_id.desc()).first()
        last = last.order_id + 1

        for panel in panels:
            orders1 = Orders1(customer_id=customer_id, panel_id=panel, order_date=str(order_date),
                               order_time=str(order_time))
            db.session.add(orders1)
            db.session.commit()

        return render_template('place_orders.html', first_name=first_name, last_name=last_name, id=customer_id,
                               panels=panels)

# new customer (must add contact info prior to orders):
else:
    Customers2 = Base.classes.customers
    customers2 = db.session.query(Customers2.customer_id).order_by(Customers2.customer_id.desc()).first()
    customer_id = customers2[0] + 1 # customer id for new customer equals last row id plus 1

    Email_Addresses2 = Base.classes.email_addresses
    new_email = Email_Addresses2(email=email)
    db.session.add(new_email)
    db.session.commit()

    Phone_Numbers2 = Base.classes.phone_numbers
    new_phone = Phone_Numbers2(phone_number=phone)
    db.session.add(new_phone)
    db.session.commit()

    Addresses2 = Base.classes.addresses
    new_address = Addresses2(street_number=street_number, street_name=street_name,
                              street_suffix=street_suffix, city=city, state=state, zip=zip_code)
    db.session.add(new_address)
    db.session.commit()

    Customers3 = Base.classes.customers
    new_customer = Customers3(first_name=first_name, last_name=last_name, date_of_birth=date_of_birth)
    db.session.add(new_customer)
    db.session.commit()

    # Create order for new customer or customer with 'changed' contact info:
    current = datetime.now()
    order_date = current.strftime('%Y-%m-%d')
    order_time = current.strftime('%H:%M:%S')

    Orders1 = Base.classes.orders

    for panel in panels: # dynamically create multiple orders from multiple panes
        orders1 = Orders1(customer_id=customer_id, panel_id=panel, order_date=order_date,
                           order_time=order_time)
        db.session.add(orders1)
        db.session.commit()

    return render_template('place_orders.html', first_name=first_name, last_name=last_name, id=customer_id,
                           panels=panels)

# CUSTOMERS VIEWS
# Customer Log-In (precedes Customer View)
@app.route('/customer_login')
def customer_login():
    return render_template('customer_login.html')

# Global
login_id = None

# CUSTOMER DASHBOARD
@app.route('/customer_dashboard', methods=['POST', 'GET'])
def customer_home():
    global login_id
    if request.method == 'GET':
        return customer_login()
    if request.method == 'POST':
        customer_id = request.form['customer_id']
        login_id = customer_id
        Customer = Base.classes.customers
        check = db.session.query(Customer.customer_id).filter_by(customer_id=login_id).count()
        first = db.session.query(Customer.first_name).filter_by(customer_id=login_id).first()
        last = db.session.query(Customer.last_name).filter_by(customer_id=login_id).first()

        if check > 0:
            return render_template('customer_dashboard.html', first=first, last=last)
        else:
            return render_template('incorrect.html',
                                   display='Not Listed: Please contact the IT Department!')

# CUSTOMER CONTACT INFO
@app.route('/contact_info')
def contact_info():
    customer_id = login_id
    # customer contact
    Customers1 = Base.classes.customers
    customers = db.session.query(Customers1).filter_by(customer_id=customer_id).first()

    Customers2 = Base.classes.customers
    address_id = db.session.query(Customers2.address_id).filter_by(customer_id=customer_id).first()

    Customers3 = Base.classes.customers

```

```

phone_id = db.session.query(Customers3.phone_id).filter_by(customer_id=customer_id).first()

Customers4 = Base.classes.customers
email_id = db.session.query(Customers4.email_id).filter_by(customer_id=customer_id).first()

Address = Base.classes.addresses
address = db.session.query(Address).filter_by(address_id=address_id[0]).first()

Email = Base.classes.email_addresses
email = db.session.query(Email).filter_by(email_id=email_id[0]).first()

Phone = Base.classes.phone_numbers
phone = db.session.query(Phone).filter_by(phone_id=phone_id[0]).first()

customer_display1 = f'{customers.first_name} {customers.last_name}'
customer_display2 = f'{address.street_number} {address.street_name} {address.street_suffix}'
customer_display3 = f'{address.city} {address.state} {address.zip}'
customer_display4 = f'{phone.phone_number}'
customer_display5 = f'{email.email}'

return render_template('contact_info.html', customer_display1=customer_display1,
                        customer_display2=customer_display2,
                        customer_display3=customer_display3,
                        customer_display4=customer_display4,
                        customer_display5=customer_display5)

@app.route('/results_form')
def results_form():
    return render_template('results_form.html')

@app.route('/results', methods=['POST', 'GET'])
def results_dashboard():
    req = request.form
    customer_id = login_id
    analytes = req.getlist('analyte_name')
    alerts = req.getlist('alert')
    start = req["start_date"]
    end = req["end_date"]
    Customer_Reports = Base.classes.customer_reports
    query1 = db.session.query(Customer_Reports).filter_by(customer_id=customer_id).filter(
        Customer_Reports.result_date.between(start, end)).filter(Customer_Reports.analyte_name.in_(analytes)).filter(
        Customer_Reports.alert.in_(alerts)).all()
    if len(query1) > 0:
        columns = []
        keys = Customer_Reports.__table__.columns
        for column in keys:
            column = f'{column}'.split(".")[1::2]
            columns.extend(column)
        data = []
        for c in query1:
            series = [c.unique_id, c.order_id, c.customer_id, c.first_name, c.last_name,
                      c.analyte_name, float(c.result), c.units_of_measure, c.result_date.strftime("%Y-%m-%d"),
                      c.result_time.strftime('%H:%M'), c.alert]
            data.append(series)

        df = pd.DataFrame(data, columns=columns)
        customer_display1 = df.to_html(index=False)
        return render_template('results_tab.html', customer_results_display=customer_display1)
    else:
        return 'You currently do not have have any results at this time!'

@app.route('/charges')
def charges_dashboard():
    customer_id = login_id
    Individual_Charges = Base.classes.customers_individual_charges
    query2 = db.session.query(Individual_Charges).filter_by(customer_id=customer_id).all()
    # customer individual charges table
    if len(query2) < 1:
        return 'You currently do not have have any charges at this time!'
    else:
        columns = []
        keys = Individual_Charges.__table__.columns
        for column in keys:
            column = f'{column}'.split(".")[1::2]
            columns.extend(column)
        data = []
        for c in query2: # c = columns
            series = [c.unique_id, c.customer_id, c.order_id, c.first_name, c.last_name,
                      float(c.panel_charge)]
            data.append(series)
        df = pd.DataFrame(data, columns=columns)
        charges_display1 = df.to_html(index=False)

        # customer grouped charges table
        Grouped_Charges = Base.classes.customer_grouped_charges
        query3 = db.session.query(Grouped_Charges).filter_by(customer_id=customer_id).all()
        columns = []
        keys = Grouped_Charges.__table__.columns
        for column in keys:
            column = f'{column}'.split(".")[1::2]
            columns.extend(column)
        data = []
        for c in query3: # c = columns
            series = [c.customer_id, c.first_name, c.last_name,
                      c.sum]
            data.append(series)
        df = pd.DataFrame(data, columns=columns)
        charges_display2 = df.to_html(index=False)

        return render_template('charges.html', charges_display1=charges_display1,

```

charges_display2=charges_display2)

```
# EMPLOYEES VIEWS
# Employee Log-In (precedes Employee Dashboard)
@app.route('/employee_login')
def employee_login():
    return render_template('employee_login.html')

# Employee Dashboard
@app.route('/employee', methods=['POST', 'GET'])
def employee_dashboard():
    if request.method == 'GET':
        return f"Access Denied: Must Log-In to View This Page!"
    if request.method == 'POST':
        employee_id = request.form['employee_id']
        # Verify employee in database
        Employees = Base.classes.employees
        employee = db.session.query(Employees).filter_by(employee_id=employee_id).count()

        if employee < 1:
            return render_template('incorrect.html',
                                   display="Not Listed: Please contact the IT Department!")
        else:
            return render_template('employee_dashboard.html', employee_display=employee_id)

# COLLECTOR VIEWS
# Collection Tubes - Displays tubes needed to be drawn per order
@app.route('/collection_tubes')
def collection_containers():
    Collector_Containers = Base.classes.collector_containers
    collector_containers_query = db.session.query(Collector_Containers).all()
    # If Query finds a matching customer in DB...
    if len(collector_containers_query) > 0:
        keys = Collector_Containers.__table__.columns
        columns = []
        for column in keys:
            column = f'"{column}"'.split('"')[1::2]
            columns.extend(column)
        data = []
        for c in collector_containers_query:
            series = [c.unique_id, c.order_id, c.customer_id, c.first_name, c.last_name,
                      c.panel_name, c.barcode_id, c.container_type, c.container_color, c.sample_type]
            data.append(series)
        df = pd.DataFrame(data, columns=columns)
        # convert floats from nulls
        bar_list = []
        for bar in df['barcode_id']:
            if "." in str(bar):
                bar = int(bar)
                bar_list.append(bar)
            else:
                bar_list.append('NULL') # Note this doesn't affect DB only used for app view
        df['barcode_id'] = bar_list

        collection_containers_display = df.to_html(index=False)
        del keys, column, columns, series, data, df # guarantees no cross-over from previous results
    else:
        collection_containers_display = 'You currently do not have have any samples to collect at this time!'

    return render_template('collection_tubes.html', collection_containers_display=collection_containers_display)

# Enter Order ID for Barcode
@app.route('/print_barcode')
def print_barcode():
    return render_template('print_barcode.html')

# Displays Barcode - Pulls from Missing Test table (test not run)
@app.route('/barcode', methods=['POST', 'GET'])
def barcode():
    if request.method == 'GET':
        return f"Access Denied: Must Log-In to View This Page!"
    if request.method == 'POST':
        order_id = request.form['order_id']
        Samples = Base.classes.samples
        samples = db.session.query(Samples).filter_by(order_id=order_id).all()
        if len(samples) < 1:
            return render_template('incorrect.html',
                                   display="You have entered an invalid or missing order id number!")
        else:
            ### get keys values
            keys = []
            columns = Samples.__table__.columns
            for column in columns:
                column = f'"{column}"'.split('"')[1::2]
                keys.extend(column)
            values = []
            for c in samples: # c = columns
                values = [int(c.barcode_id), c.order_id, c.employee_id, c.customer_id,
                          c.collection_date.strftime("%Y-%m-%d"), c.collection_time.strftime('%H:%M')]
            # create dict to use to display k,v pairs in html
            display_dict = {}
            for k, v in zip(keys, values):
                display_dict[k] = v
            # create the barcode and store in static folder
            qr = qrcode.QRCode(version=1,
                                error_correction=qrcode.constants.ERROR_CORRECT_L,
                                box_size=10,
                                border=2)
```

```

qr.add_data(int(order_id))
qr.make(fit=True)

img = qr.make_image(fill_color="black", back_color="white")
root = os.path.dirname(os.getcwd())
path = r'app\static\barcode.png'
img.save(path)
display_dict = display_dict

    return render_template('barcode.html', display_dict=display_dict)

# TECH VIEWS

# customer results
@app.route('/customer_results')
def customer_results():
    return render_template('customer_results.html')

# Summary of Patient Range Stats
@app.route('/customer_results_table', methods=['POST', 'GET'])
def customer_results_table():
    if request.method == 'GET':
        return f"Access Denied: Must Log-In to View This Page!"
    if request.method == 'POST':
        req1 = request.form['order_id']
        req2 = request.form['customer_id']
        if len(req1) == 0 and len(req2) == 0:
            customer_results_display = 'Please go back and make a selection!'
        elif len(req1) != 0 and len(req2) != 0:
            customer_results_display = 'Please enter only one field!'
        else:
            if len(req1) > 0:
                Customer_Ranges = Base.classes.customer_reports
                query1 = db.session.query(Customer_Ranges).filter_by(order_id=int(req1)).all()
                if len(query1) == 0:
                    return 'You have entered an invalid ID, Please Try Again!'
                else:
                    columns = []
                    keys = Customer_Ranges.__table__.columns
                    for column in keys:
                        column = f'{column}'.split(".")[1::2]
                        columns.extend(column)
                    data = []
                    for c in query1:
                        series = [c.unique_id, c.order_id, c.customer_id, c.first_name, c.last_name,
                                c.analyte_name, float(c.result), c.units_of_measure,
                                c.result_date.strftime("%Y-%m-%d"),
                                c.result_time.strftime('%H:%M'), c.alert]
                        data.append(series)
                    df = pd.DataFrame(data, columns=columns)
                    customer_results_display = df.to_html(index=False)
                    del keys, column, columns, series, data, df # guarantees no cross-over from previous results
            else:
                Customer_Ranges = Base.classes.customer_reports
                query2 = db.session.query(Customer_Ranges).filter_by(customer_id=int(req2)).all()
                if len(query2) > 0:
                    columns = []
                    keys = Customer_Ranges.__table__.columns
                    for column in keys:
                        column = f'{column}'.split(".")[1::2]
                        columns.extend(column)
                    data = []
                    for c in query2:
                        series = [c.unique_id, c.order_id, c.customer_id, c.first_name, c.last_name,
                                c.analyte_name, float(c.result), c.units_of_measure,
                                c.result_date.strftime("%Y-%m-%d"),
                                c.result_time.strftime('%H:%M'), c.alert]
                        data.append(series)
                    df = pd.DataFrame(data, columns=columns)
                    customer_results_display = df.to_html(index=False)
                    del keys, column, columns, series, data, df # guarantees no cross-over from previous results

    return render_template('customer_results_tab.html', customer_results_display=customer_results_display)

# Patient Range Form to Select Attributes
@app.route('/patient_ranges')
def customer_ranges_home():
    return render_template('patient_ranges.html')

# Summary of Patient Range Stats
@app.route('/patient_ranges_tab', methods=['POST', 'GET'])
def customer_ranges_table():
    req = request.form
    analytes = req.getlist('analyte_name')
    Patient_Ranges = Base.classes.manager_range_report
    query = db.session.query(Patient_Ranges).filter(Patient_Ranges.analyte_name.in_(analytes)).all()
    columns = []
    keys = Patient_Ranges.__table__.columns
    for column in keys:
        column = f'{column}'.split(".")[1::2]
        columns.extend(column)
    data = []
    for c in query:
        series = [c.analyte_name, c.n, c.mean_actual, c.sd_actual,
                  c.mean_target, c.sd_target, c.make, c.model]
        data.append(series)
    df = pd.DataFrame(data, columns=columns)
    patient_ranges_tab = df.to_html(index=False)
    del keys, column, columns, series, data, df

```

```

return render_template('patient_ranges_tab.html', analytes=analytes, patient_ranges_tab=patient_ranges_tab)

#####
@app.route('/moving_averages')
def moving_averages():
    return render_template('moving_averages.html')

@app.route('/moving_averages_table', methods=['POST', 'GET'])
def moving_averages_tab():
    req = request.form
    if len(req) < 1:
        return 'Please make a select!'
    else:
        analytes_ = req.getlist('analyte_name')
        start = req["start_date"]
        end = req["end_date"]
        Results = Base.classes.customer_reports
        query = db.session.query(Results).filter(Results.analyte_name.in_(analytes_)).filter(
            Results.result_date.between(start, end)).all()

        if len(query) < 0:
            return 'Sorry no results for this selection!'
        else:
            columns = []
            keys = Results.__table__.columns
            for column in keys:
                column = f'{column}'.split(".")[1::2]
                columns.extend(column)
            data = []
            for c in query:
                series = [c.unique_id, c.order_id, c.customer_id, c.first_name, c.last_name,
                           c.analyte_name, float(c.result), c.units_of_measure,
                           c.result_date.strftime("%Y-%m-%d"),
                           c.result_time.strftime('%H:%M'), c.alert]
                data.append(series)
            df = pd.DataFrame(data, columns=columns)

            # descriptive statistics
            descriptive = pd.pivot_table(df, values=['result'], index=['analyte_name', 'result_date'],
                                         aggfunc={np.mean, np.std, np.min, np.max}).round(2)
            moving_averages_display1 = descriptive.to_html(index=False)

            # smooth moving averages per analyte
            SMA = df.loc[:, ['result_date', 'analyte_name', 'result']]
            SMA['Smooth Moving Average'] = SMA['result'].rolling(5).mean()
            SMA = SMA.loc[:, ['result_date', 'analyte_name', 'Smooth Moving Average']]
            cols = ['analyte_name', 'result_date']
            SMA = SMA.sort_values(by=cols)
            moving_averages_display2 = SMA.to_html(index=False)

            # z_SCORES means vs. expect mean (population - manufacturer)
            # obtain EXPECTED MEANS from database
            Tests = Base.classes.analytes
            query2 = db.session.query(Tests).filter(Tests.analyte_name.in_(analytes_)).all()
            columns = []
            keys = Tests.__table__.columns
            for column in keys:
                column = f'{column}'.split(".")[1::2]
                columns.extend(column)
            data1 = []
            for a in query2:
                series = [a.analyte_id, a.serial_id, a.panel_id, a.analyte_name,
                           a.analyte_mean, a.analyte_sd, a.units_of_measure]
                data1.append(series)
            df1 = pd.DataFrame(data1, columns=columns)
            df1 = df1.loc[:, ['analyte_name', 'analyte_mean', 'analyte_sd']]
            # df2 = pd.pivot_table(df1, values=['result'], index=['analyte_name'], aggfunc={np.mean, np.std}).round(2)
            df2 = df1.groupby(df1['analyte_name']).result.agg(["mean", "std"]).round(2)
            combined = merged = pd.merge(df1, df2, on='analyte_name', how="inner")
            combined['z_score'] = combined['mean'].astype(float) - combined['analyte_mean'].astype(float)
            combined['p_value'] = combined['z_score'].applymap(lambda x: scipy.stats.t.sf(abs(x), 1))
            moving_averages_display3 = combined.to_html(index=False)

        return render_template('moving_averages_tab.html', analytes=analytes_,
                               moving_averages_tab2=moving_averages_display2,
                               moving_averages_tab3=moving_averages_display3)

#####
# Missing tests
@app.route('/missing_tests')
def missing_tests():
    Missing_Tests = Base.classes.missing_tests_report
    missing_tests_query = db.session.query(Missing_Tests).all()
    # If Query finds a matching customer in DB...
    if len(missing_tests_query) > 0:
        keys = Missing_Tests.__table__.columns
        columns = []
        for column in keys:
            column = f'{column}'.split(".")[1::2]
            columns.extend(column)
        data = []
        for c in missing_tests_query:
            series = [c.order_id, c.customer_id, c.analyte_name, c.result,
                       c.explanation]
            data.append(series)
        df = pd.DataFrame(data, columns=columns)
        missing_tests_display = df.to_html(index=False)
        del keys, column, columns, series, data, df # guarantees no cross-over from previous results
    else:

```

```

missing_tests_display = 'You currently do not have any results at this time!'

return render_template('missing_tests.html', missing_tests_display=missing_tests_display)

# MANAGER VIEWS
# QC INDIVIDUAL RESULTS
@app.route('/qc_values')
def qc_values_home():
    return render_template('qc_values.html')

@app.route('/qc_values_tab', methods=['POST', 'GET'])
def qc_values_tab():
    req = request.form
    start = req["start_date"]
    end = req["end_date"]
    if start > end:
        qc_values_tab = 'Please choose a start date prior to an end date!'
    else:
        analytes = req.getlist('analyte_name')
        QC_Values = Base.classes.manager_individual_qc_report
        query = db.session.query(QC_Values).filter(QC_Values.analyte_name.in_(analytes)).filter(
            QC_Values.qc_date.between(start, end)).all()
        columns = []
        keys = QC_Values.__table__.columns
        for column in keys:
            column = f"{column}".split(".")[1::2]
            columns.extend(column)
        data = []
        for c in query:
            series = [c.unique_id, c.qc_date, c.qc_time, c.qc_level, c.analyte_name, c.qc_value, c.units_of_measure,
                      c.make, c.model, c.alert]
            data.append(series)
        df = pd.DataFrame(data, columns=columns)
        qc_values_tab = df.to_html(index=False)
        del keys, column, columns, series, data, df

    return render_template('qc_values_tab.html', qc_values_tab=qc_values_tab)

# QC CUMULATIVE RESULTS
@app.route('/qc_cumulative')
def qc_individual_home():
    return render_template('qc_cumulative.html')

@app.route('/qc_cumulative_tab', methods=['POST', 'GET'])
def qc_individual_tab():
    req = request.form
    start = request.form["start_date"]
    end = request.form["end_date"]
    if start > end:
        qc_cumulative_tab = 'Please choose a start date prior to an end date!'
    else:
        analytes = req.getlist('analyte_name')
        QC_Values = Base.classes.manager_individual_qc_report
        query = db.session.query(QC_Values).filter(QC_Values.analyte_name.in_(analytes)).filter(
            QC_Values.qc_date.between(start, end)).all()
        columns = []
        keys = QC_Values.__table__.columns
        for column in keys:
            column = f"{column}".split(".")[1::2]
            columns.extend(column)
        data = []
        for c in query:
            series = [c.unique_id, c.qc_date, c.qc_time, c.qc_level, c.analyte_name, float(c.qc_value),
                      c.units_of_measure, c.make, c.model, c.alert]
            data.append(series)
        df = pd.DataFrame(data, columns=columns)
        df = df.loc[:, ['qc_date', 'qc_level', 'analyte_name', 'qc_value']]
        df = df.pivot_table(values='qc_value', index=['analyte_name', 'qc_level'], aggfunc=['mean', 'std']).round(1)
        df = df.reset_index()
        qc_cumulative_tab = df.to_html(index=False)
        del keys, column, columns, series, data, df

    return render_template('qc_cumulative_tab.html', analytes=analytes, qc_cumulative_tab=qc_cumulative_tab)

# QC CURRENT SUMMARY
@app.route('/qc_summary')
def qc_grouped():
    QC_Summary = Base.classes.manager_grouped_qc_report
    qc_summary_query = db.session.query(QC_Summary).all()
    # If Query finds a matching customer in DB...
    if len(qc_summary_query) > 0:
        keys = QC_Summary.__table__.columns
        columns = []
        for column in keys:
            column = f"{column}".split(".")[1::2]
            columns.extend(column)
        data = []
        for c in qc_summary_query:
            series = [c.unique_id, c.qc_level, c.analyte_name, float(c.mean),
                      float(c.sd), c.units_of_measure, c.make, c.model]
            data.append(series)
        df = pd.DataFrame(data, columns=columns)
        qc_summary_display = df.to_html(index=False)
        del keys, column, columns, series, data, df # guarantees no cross-over from previous results
    else:
        qc_summary_display = 'You currently do not have any results at this time!'

    return render_template('qc_summary.html', qc_summary_display=qc_summary_display)

```

```

# QC VISUALS

# QC SCATTER PLOT FORM
@app.route('/qc_scatter')
def qc_scatter_home():
    return render_template('qc_scatter.html')

# QC SCATTER PLOT
@app.route('/qc_scatter_graph', methods=['POST', 'GET'])
def qc_scatter_plot():
    req = request.form
    start = request.form["start_date"]
    end = request.form["end_date"]
    if start > end:
        return 'Please choose a start date prior to an end date!'
    else:
        analytes = req.getlist('analyte_name')
        QC_Values = Base.classes.manager_individual_qc_report
        query = db.session.query(QC_Values).filter(QC_Values.analyte_name.in_(analytes)).filter(
            QC_Values.qc_date.between(start, end)).all()
        columns = []
        keys = QC_Values.__table__.columns
        for column in keys:
            column = f'"{column}".split(".")[1::2]'
            columns.extend(column)
        data = []
        for c in query:
            series = [c.unique_id, c.qc_date, c.qc_time, c.qc_level, c.analyte_name, float(c.qc_value),
                      c.units_of_measure, c.make, c.model, c.alert]
            data.append(series)
        df = pd.DataFrame(data, columns=columns)
        df1 = df[(df['analyte_name'] == analytes[0]) & (df['qc_level'] == str(1))]
        df2 = df[(df['analyte_name'] == analytes[0]) & (df['qc_level'] == str(2))]
        df3 = df[(df['analyte_name'] == analytes[0]) & (df['qc_level'] == str(3))]
        y1 = df1['qc_value']
        y2 = df2['qc_value']
        y3 = df3['qc_value']

        fig, ax = plt.subplots(3, 1, sharex=False, figsize=(12, 8),
                               constrained_layout=True)

        fig.suptitle(analytes[0])

        qc1 = sns.scatterplot(data=df1, x='qc_date', y='qc_value', ax=ax[0])
        qc1.set_title(f'{analytes[0]} QC LEVEL 1')
        qc1.axhline(y1.mean() + (2 * y1.std()),
                    color='green',
                    lw=3,
                    alpha=0.7)
        qc1.axhline(y1.mean() - (2 * y1.std()),
                    color='green',
                    lw=3,
                    alpha=0.7)

        qc2 = sns.scatterplot(data=df2, x='qc_date', y='qc_value', ax=ax[1])
        qc2.set_title(f'{analytes[0]} QC LEVEL 2')
        qc2.axhline(y2.mean() + (2 * y2.std()),
                    color='green',
                    lw=3,
                    alpha=0.7)
        qc2.axhline(y2.mean() - (2 * y2.std()),
                    color='green',
                    lw=3,
                    alpha=0.7)

        qc3 = sns.scatterplot(data=df3, x='qc_date', y='qc_value', ax=ax[2])
        qc3.set_title(f'{analytes[0]} QC LEVEL 3')
        qc3.axhline(y3.mean() + (2 * y3.std()),
                    color='green',
                    lw=3,
                    alpha=0.7)
        qc3.axhline(y3.mean() - (2 * y3.std()),
                    color='green',
                    lw=3,
                    alpha=0.7)

        obj = io.BytesIO()
        fig.savefig(obj, format='png')
        get = obj.getbuffer()
        encoded = base64.b64encode(obj.getbuffer()).decode("utf-8")
        string1 = base64.b64encode(obj.getbuffer()).decode("utf-8")

        return f"<img src='data:image/png;base64,{string1}'/>"

# QC WESTGARD RULES
@app.route('/westgard')
def westgard_home():
    return render_template('westgard.html')

@app.route('/westgard_table', methods=['POST', 'GET'])
def westgard_table():
    req = request.form
    analyte = req.getlist('analyte_name')
    if len(analyte) < 1:
        return 'Please Make a Selection!'
    else:
        # Create dataframe for qc_analytes table
        QC_Analytes = Base.classes.qc_analytes

```



```

qc_analytes = db.session.query(QC_Analytes).all()
columns1 = []
keys = QC_Analytes.__table__.columns
for column in keys:
    column = f'"{column}'.split(".")[1::2]
    columns1.extend(column)
data1 = []
for c in qc_analytes:
    series = [c.qc_analyte_id, c.qc_panel_id, c.qc_level, c.manager_id, float(c.qc_range_low),
              float(c.qc_mean), float(c.qc_range_low)]
    data1.append(series)
qc_analytes_df = pd.DataFrame(data1, columns=columns1)


# Create dataframe for qc_values table
QC_Values = Base.classes.qc_values
qc_values = db.session.query(QC_Values).all()
columns2 = []
keys = QC_Values.__table__.columns
for column in keys:
    column = f'"{column}'.split(".")[1::2]
    columns2.extend(column)
data2 = []
for c in qc_values:
    series = [c.qc_value_id, c.qc_analyte_id, float(c.qc_value), c.qc_date, c.qc_time]
    data2.append(series)
qc_values_df = pd.DataFrame(data2, columns=columns2)

# Create dataframe for qc_analytes table
Analytes = Base.classes.analytes
analytes = db.session.query>Analytes).filter>Analytes.analyte_name.in_(analyte)).all()
columns3 = []
keys = Analytes.__table__.columns
for column in keys:
    column = f'"{column}'.split(".")[1::2]
    columns3.extend(column)
data3 = []
for c in analytes:
    series = [c.analyte_id, c.serial_id, c.panel_id, c.analyte_name, float(c.analyte_mean),
              float(c.analyte_sd), c.units_of_measure]
    data3.append(series)
analytes_df = pd.DataFrame(data3, columns=columns3)

# merge on foreign key qc_analyte_id
merge = pd.merge(qc_analytes_df, qc_values_df, how='left', on='qc_analyte_id')
merge = merge.drop(['manager_id', ], axis=1)
# to_numpy silences new warning due to change in ufunc behavior
merge[['Deviation']] = np.subtract(merge[['qc_value']], merge[['qc_mean']].to_numpy())
merge[['1SD']] = np.subtract(merge[['qc_range_high']], merge[['qc_mean']].to_numpy())
merge[['1SD']] = np.abs(np.divide(merge[['1SD']], 2))
merge[['SDI']] = np.divide(merge[['Deviation']], merge[['1SD']].to_numpy())
zero_df = pd.DataFrame([{'SDI': 0}])
# add zero to beginning since day1 cant have a difference
pre = pd.concat([zero_df, merge[['SDI']][1:]], axis=0).reset_index(drop=True)
post = pd.concat([zero_df, merge[['SDI']][:-1]], axis=0).reset_index(drop=True)
merge[['DeltaSDI']] = post - pre
# Creating a Range 4SD (subsequent results 4SD apart)
merge[['Range4SD Rule']] = merge[['DeltaSDI']].applymap(lambda s: "VIOLATION" if (s > 4 or s < -4) else "OK")
# Create a 3 SD rule - any results > or < 3SD = violation
merge[['>3SD Rule']] = merge[['SDI']].applymap(lambda r: "VIOLATION" if (r < -3 or r > 3) else "OK")
# Create a > 2SD outlier column
merge[['>2SD Outlier']] = merge[['SDI']].applymap(lambda r: "VIOLATION" if (r < -2 or r > 2) else "OK")
# Creating a 2-2SD rule column - any two subsequent results > or < (+/-) 2SD but less than (+/-)3SD respectively
pre_list = pre.iloc[:, 0].tolist()
post_list = post.iloc[:, 0].tolist()
outcomes = []
for pr, po in zip(pre_list, post_list):
    if (pr > 2 and po > 2 and pr < 3 and po < 3) or (pr < -2 and po < -2 and pr > -3 and po > -3):
        outcomes.append("VIOLATION")
    else:
        outcomes.append("OK")
merge[['2-2SD Rule']] = pd.Series([outcomes])
# note a qc_panel_id, although confusingly named, corresponds to 1 analyte
# qc_panel_id is PK for qc_panel which maps an analyte to a panel to clarify
final = pd.merge(analytes_df, merge, left_on='analyte_id', right_on='qc_panel_id')
final = final[['qc_date', 'qc_time', 'analyte_name', 'qc_value', 'qc_mean', '1SD', 'SDI',
               'DeltaSDI', '>2SD Outlier', '2-2SD Rule', '>3SD Rule', 'Range4SD Rule']]
display2 = final.to_html(index=False)
# Finally, create a table that gives the viewer definitions of each rule:
r_sdi = "Standard Deviation Index (Standard Deviation Units)"
r_delta = "Difference in SDI from previous run"
r_2sd = "All values with SDI > 2 are considered and outlier"
r_22sd = "If two subsequent runs are: 2SD>SDI<3SD or -2SD>SDI<-3SD in a row"
r_3sd = "If any value is > (+/-)3SD"
r_r4s = "If two subsequent values are > (+/-)4SD apart"
index = ['SDI', 'DeltaSDI', '>2SD Outlier', '2-2SD Rule', '>3SD Rule', 'Range4SD Rule']
definitions = [r_sdi, r_delta, r_2sd, r_22sd, r_3sd, r_r4s]
columns_ = ["-----Definitions-----"]
rules_df = pd.DataFrame(definitions, index=index, columns=columns_)
display1 = rules_df.to_html(index=False)

return render_template('westgard_tab.html', analytes=analytes[0],
                      display1=display1,
                      display2=display2)

```


In []:  <insert in any .png file you wish?

in template folder (lives inside app folder)

In []:  about.html

```
{% extends "parent_templates/parent.html" %}
{% block title %}QC Summary Statistics{% endblock %}


{% block main %}
<br><br><br><br>
<p>A clinical reference laboratory is a private laboratory that allows customers to submit <br>
orders online for medical laboratory tests commonly referred to as ‘blood tests’. Some <br>
examples are companies such as Quest@3 or Any-Lab-Test-Now@2 where customers may <br>
order lab tests online, have their blood drawn, and test results are then produced for <br>
laboratory tests that customers pay for online.</p>

<p>This application hopes to mimic such aforementioned company web pages by strategically <br>
implementing great SQL database DDL and DML design not limited to procedures, triggers, <br>
constraints, and schema design. The web page utilizes the power of Python's Flask library <br>
to incorporate the database in as an object-oriented representation of a relational database.</p>

<p>Once, the database is instantiated with Flask, any number of Python modules can be integrated <br>
to provide any number of functionalities</p>

<p>The main focus of this database was solely on the integration of Flask with SQL database and <br>
the use of proper database DDL and DML techniques. This database is purposely in no way <br>
meant to suffice for security or a high level of database design.</p>

{% endblock %}
```

In []:  barcode.html

```
{% extends "parent_templates/employee.html" %}
{% block title %}Barcode{% endblock %}


{% block main %}

<style type="text/css">
  ul {
    list-style-type: none;
  }
</style>

<head>
  <meta charset="UTF-8">
  <title>Barcode</title>
</head>

<body>
<br>
<h2>Customer Barcode:</h2>
<br>
  <img src = 'static/barcode.png' >
  <ul>
    {% for k,v in display_dict.items() %}
    <li><b>{{ k }}: {{ v }}</b></li>
    {% endfor %}
  </ul>
</body>

{% endblock %}
```

In []:  charges.html

```
{% extends "parent_templates/customer.html" %}
{% block title %}Billing{% endblock %}

{% block main %}
<head>
  <meta charset="UTF-8">
  <title>Customer_Dashboard</title>
</head>
<body>
<br><br>
<h2>Charges:</h2>

<p>Individual charges are a list of charges per order and update after a new order</p>

<ul>
<b>INDIVIDUAL CHARGES:</b><br>
  {{charges_display1 | safe}}
</ul>
<br>

<p>The sum value in this table should equal the total sum of all individual charges</p>

<ul>
<b>TOTAL CHARGES:</b><br>
  {{charges_display2 | safe}}
</ul>

</body>

{% endblock %}
```

In []:

collect_barcode.html

```
{% extends "parent_templates/employee.html" %}
{% block title %}QC Summary Statistics{% endblock %}

{% block main %}

<style type="text/css">
  ul {
    list-style-type: none;
  }
</style>

<head>
  <meta charset="UTF-8">
  <title>Barcode</title>
</head>

<body>
<br>
<h2>Customer Barcode:</h2>
<br>
  <img src = '/static/barcode.png' >
  <ul>
    {% for k,v in display_dict.items() %}
    <li><b>{{ k }}: {{ v }}</b></li>
    {% endfor %}
  </ul>
</body>

{% endblock %}
```

In []:

collection_tubes.html

```
{% extends "parent_templates/employee.html" %}
{% block title %}Collection Tubes{% endblock %}

{% block main %}
<head>
  <meta charset="UTF-8">
  <title>Collection_Tubes</title>
</head>
<body>
<ul>
  <br><br>
<h2>COLLECTION TUBES NEEDED TO BE DRAWN:</h2> <br>
  {{collection_containers_display | safe}}
</ul>

</body>

{% endblock %}
```

In []:

contact_info.html

```
{% extends "parent_templates/customer.html" %}
{% block title %}Customer Dashboard{% endblock %}

{% block main %}
<head>
  <meta charset="UTF-8">
  <title>Customer_Dashboard</title>
</head>
<body>
<br><br>
<h2>Customer Dashboard:</h2>
<br><br>

<br>
<p>Welcome to the Customer Dashboard! Please make a selection above to view current results or charges</p>

<ul>
<b>Welcome!</b> <br>
  {{customer_display1 | safe}}
</ul>
<br>
<ul>
<b>Current Address:</b> <br>
  {{customer_display2 | safe}}
  {{customer_display3 | safe}}
</ul>
<br>
<ul>
<b>Current Phone:</b> <br>
  {{customer_display4 | safe}}
</ul>

<ul>
<b>Current Email:</b> <br>
  {{customer_display5 | safe}}
</ul>

</body>

{% endblock %}
```

In []:

customer_dashboard.html

```
{% extends "parent_templates/customer.html" %}

{% block title %}Customer Dashboard{% endblock %}

{% block main %}
<body>
<h2>Welcome, currently login as:</h2>
<b>{{first[0]}} {{last[0]}} </b>
</body>

{% endblock %}
```

In []: `customer_login.html`

```
{% extends "parent_templates/parent.html" %}

{% block title %}Query{% endblock %}

{% block main %}
<div class="container">
  <div class="row">
    <div class="col"></div>
    <title>Text alignment</title>
    <style>
      h1{text-align:left;}
    </style>
    <body>

    </body>
  </div>
</div>
<br><br>
<h2>Customer Log-In:</h2>
<br><br>
<p>To begin, simply type any customer id between 100000 and 100499.</p>
<p>or</p>
<p>You may also use any new customer id obtained from after placing an order </p>
<form action="/customer_dashboard" method = "POST">
  <p>Customer ID <input type = "text" name = "customer_id" /></p>
  <p><input type = "submit" value = "Submit" /></p>
</form>

<p>Please note to view different patients simply submit a different customer id number</p>
<p>Once "logged-in" you will remain in the customer dashboard indefinitely until a new id is entered</p>
<p>Please note the purpose of this application is solely to demonstrate database functionality</p>
<p>This application by no means resemble any form of cybersecurity in a modern application</p>
<p>I hope to work on that in my next personal project</p>
{% endblock %}
```

In []: `customer_results.html`

```
{% extends "parent_templates/employee.html" %}
{% block title %}Customer Results{% endblock %}

{% block main %}

<h2>Customer Results:</h2>
<br><br>
<h4><b>Enter a Customer or Order ID Below (only 1 entry accepted): </b></h4>

<form action="/customer_results_table" method = "POST">
  <p>order_id <input type = "text" name = "order_id" /></p>
  <p><b>OR</b></p>
  <p>customer_id <input type = "text" name = "customer_id" /></p>
  <p><input type = "submit" value = "Submit" /></p>
</form>
<p>Note:Due to HIPPA restrictions results are limited to one order or customer id number!</p>

{% endblock %}
```

In []: `customer_results_tab.html`

```
{% extends "parent_templates/employee.html" %}
{% block title %} Customer Results {% endblock %}

{% block main %}
<head>
  <meta charset="UTF-8">
  <title>Customer Results</title>
</head>
<body>
<br><br>
<h2>Customer Results:</h2> <br>
{{customer_results_display | safe}}

</body>

{% endblock %}
```

In []: `employee_dashboard.html`

```
{% extends "parent_templates/employee.html" %}

{% block title %}Employee Dashboard{% endblock %}

{% block main %}
```

```

<body>
<br><br>
<h2>Welcome, currently login as:</h2>
<h2>{{employee_display}}</h2>
<br>
<br>
<p>Note: to move around use links are your browsers arrows (exiting may log out user)</p>
</body>

{% endblock %}

```

In []: `employee_login.html`

```

{% extends "parent_templates/parent.html" %}

{% block title %}Employee Log-In{% endblock %}

{% block main %}
<div class="container">
  <div class="row">
    <div class="col"></div>
    <title>Text alignment</title>
    <style>
      h1{text-align:left;}
    </style>
    <body>

    </body>

  </div>
</div>
<br><br>
<h2>Employee Log-In:</h2>
<br><br>
<form action="/employee" method = "POST">
  <p>Employee ID <input type = "text" name = "employee_id" /></p>
  <p><input type = "submit" value = "Submit" /></p>
</form>
<br><br>
<p>Choose and employee ID between 100000 and 100039</p>
{% endblock %}

```

In []: `incorrect.html`

```

{% extends "parent_templates/parent.html" %}
{% block title %}Results{% endblock %}

{% block main %}
<head>
  <meta charset="UTF-8">
  <title>Customer_Dashboard</title>
</head>
<body>
<ul>
  <br><br>
<b> {{display | safe}} </b>

</ul>
</body>

{% endblock %}

```

In []: `index.html`

```

{% extends "parent_templates/parent.html" %}

{% block title %}Home Page{% endblock %}

{% block main %}

<h1>Welcome to Fake-Lab!</h1>
<br><br>
<p>Please make a selection above.</p>
<p> Further instructions for each link above are provided on the landing page.</p>
<p> Information on the project itself is located in the 'About' Page</p>
<br>
<br>
<h4>NOTE: ALL PERSONS, DATA, PLACES, AND ENTITIES ARE FICTITIOUS AND HAVE BEEN SYNTHESIZED!</h4>
{% endblock %}

```

In []: `missing_tests.html`

```

{% extends "parent_templates/employee.html" %}
{% block title %} Missing Tests {% endblock %}

{% block main %}
<head>
  <meta charset="UTF-8">
  <title>Missing Tests</title>
</head>
<body>
<ul>
  <br><br>
<h2>The Following Test Need To Be Run:</h2> <br>
  {{missing_tests_display | safe}}
</ul>

```

```
</body>
```

```
{% endblock %}
```

In []: `moving_average.html`

```
{% extends "parent_templates/employee.html" %}
{% block title %}Quality Control Cumulative Statistics{% endblock %}

{% block main %}
<body>
<br>
<h2>Patient Results Selected Statistics</h2>
<br>
<form action="/moving_averages_table" method = "POST">
  <p><b>Select a start and end date:</b></p>
  <label for="start"><b>Start date:</b></label>
  <input type="date" id="start" name="start_date"
    value="2022-03-01">
    <br><br><br>
  <label for="end"><b>End date:</b></label>
  <input type="date" id="end" name="end_date"
    value="2022-08-08">
    <br><br><br><br>
  <label for="analyte_name"><b>Choose an analyte:</b></label>
  <br>
  <select name="analyte_name" id="analyte_name" >
    <option value="SODIUM" >SODIUM</option>
    <option value="POTASSIUM" >POTASSIUM</option>
    <option value="BICARBONATE" >BICARBONATE</option>
    <option value="CALCIUM" >CALCIUM</option>
    <option value="CHLORIDE" >CHLORIDE</option>
    <option value="GLUCOSE" >GLUCOSE</option>
    <option value="PROTEIN" >PROTEIN</option>
    <option value="CREATININE" >CREATININE</option>
    <option value="GFR" >GFR</option>
    <option value="UREA NITROGEN" >UREA NITROGEN</option>
    <option value= "2HRGTT" > 2HRGTT </option>
    <option value="A1C" >A1C </option>
    <option value="AST" >AST </option>
    <option value="ALT" >ALT </option>
    <option value="GGT" >GGT </option>
    <option value="ALBUMIN" >ALBUMIN</option>
    <option value="CHOLESTEROL" >CHOLESTEROL</option>
    <option value="TRIGLYCERIDE" >TRIGLYCERIDE</option>
    <option value="LDL" >LDL </option>
    <option value="HDL" >HDL</option>
    <option value="WBC" >WBC </option>
    <option value="RBC " >RBC </option>
    <option value="HEMOGLOBIN" >HEMOGLOBIN</option>
    <option value="HEMATOCRIT" >HEMATOCRIT</option>
    <option value="PLATELET" >PLATELET</option>
    <option value="PROTIME" >PROTIME</option>
    <option value="APTT" >APTT</option>
  </select>
  <br><br>
  <label for="analyte_name"><b>Choose an analyte:</b></label>
  <br>
  <input type="submit" value="Submit">
</form>

<br><br>
</body>

{% endblock %}
```

In []: `moving_averages_tab.html`

```
{% extends "parent_templates/employee.html" %}
{% block title %}Patient Moving Averages{% endblock %}

{% block main %}

<style type="text/css">
  ul {
    list-style-type: none;
  }
</style>
<head>
  <meta charset="UTF-8">
  <h2>Patient Moving Averages For:</h2>
  <h3>{{analytes[0]}}</h3>
</head>
<body>

  <br><br>
  <h2>Moving Averages by Analyte:</h2> <br>
  {{moving_averages_tab2 | safe}}
  <br>

  <br><br>
  <b>z_score and p_value by Analyte:</b> <br>
  {{moving_averages_tab3 | safe}}
  <br>

</body>

{% endblock %}
```

```
{% extends "parent_templates/parent.html" %}

{% block title %}Place an Order{% endblock %}

{% block main %}
    <body>
        <h2>Welcome to Fake-Lab:</h2>
        <br><br>
        <h3>Fill Out Customer Information Below:</h3>
        <br><br>
    </body>

    <p><b>Personal Information:</b></p>

    <br>
    <form action="/place_order" method = "POST">
        <p>First Name: <input type ="text" name = "first_name" /></p>
        <p>Last Name: <input type ="text" name = "last_name" /></p>
        <p>Date of Birth:<input type ="date" name = "date_of_birth" /></p>
        <p><b>Contact Information:</b></p>
        <p>Street Number(Any Numerals) <input type ="text" name = "street_number" /></p>
        <p>Street Name (Any Characters) <input type ="text" name = "street_name" /></p>
        <p>Street Suffix <input type ="text" name = "street_suffix" /></p>
        <p>City (Any Characters)<input type ="text" name = "city" /></p>
        <p>State (Enter Characters) <input type ="text" name = "state" /></p>
        <p>Zip (Enter #####)<input type="text" name = "zip" /></p>
        <p>Email(Enter anything + @fakemail.fake) <input type="email" name = "email" /></p>
        <p>Phone(Enter 555-555-####) <input type="text" name = "phone" /></p>

        <br><br>
        <h3>Choose One or More Test Panels Below:</h3>
        <br><br>
        <div>
            <input type="checkbox" id="100000" name="100000">
            <label for="100000">BMP PANEL PANEL $29.99</label>
        </div>
        <div>
            <input type="checkbox" id="100001" name="100001">
            <label for="100001">LIVER PANEL $34.49</label>
        </div>
        <div>
            <input type="checkbox" id="100002" name="100002">
            <label for="100002">RENAL PANEL $25.50</label>
        </div>
        <div>
            <input type="checkbox" id="100003" name="100003">
            <label for="100003">DIABETES PANEL $45.00</label>
        </div>
        <div>
            <input type="checkbox" id="100004" name="100004">
            <label for="100004">LIPID PANEL $55.00</label>
        </div>
        <div>
            <input type="checkbox" id="100005" name="100005">
            <label for="100005">CBC PANEL $24.99</label>
        </div>
        <div>
            <input type="checkbox" id="100006" name="100006">
            <label for="100006">COAG PANEL $59.99</label>
        </div>
        <br>
        <input type="submit" value="Place Order">
    </form>

<br><br>
    <p>Please wait a few seconds for form to submit</p>
{% endblock %}
```

```
{% extends "parent_templates/employee.html" %}
{% block title %}Customer Ranges Summary Statistics{% endblock %}

{% block main %}
<body>

<h2>Customer Ranges Summary Statistics</h2>
    <form action="/patient_ranges_tab" method = "POST">
        <label for="analyte_name"><b>Choose an analyte:</b></label>
        <br>
        <select name="analyte_name" id="analyte_name" multiple>
            <option value="SODIUM" selected>SODIUM</option>
            <option value="POTASSIUM" selected>POTASSIUM</option>
            <option value="BICARBONATE" selected>BICARBONATE</option>
            <option value="CALCIUM" selected>CALCIUM</option>
            <option value="CHLORIDE" selected>CHLORIDE</option>
            <option value="GLUCOSE" selected>GLUCOSE</option>
            <option value="PROTEIN" selected>PROTEIN</option>
            <option value="CREATININE" selected>CREATININE</option>
            <option value="GFR" selected>GFR</option>
            <option value="UREA NITROGEN" selected>UREA NITROGEN</option>
            <option value="2HRGTT">2HRGTT selected</option>
            <option value="A1C" selected>A1C </option>
            <option value="AST" selected>AST </option>
            <option value="ALT" selected>ALT </option>
            <option value="GGT" selected>GGT </option>
            <option value="ALBUMIN" selected>ALBUMIN</option>
            <option value="CHOLESTEROL" selected>CHOLESTEROL</option>
            <option value="TRIGLYCERIDE" selected>TRIGLYCERIDE</option>
```

```

<option value="LDL" selected>LDL </option>
<option value="HDL" selected>HDL</option>
<option value="WBC" selected>WBC </option>
<option value="RBC selected">RBC </option>
<option value="HEMOGLOBIN" selected>HEMOGLOBIN</option>
<option value="HEMATOCRIT" selected>HEMATOCRIT</option>
<option value="PLATELET" selected>PLATELET</option>
<option value="PROTIME" selected>PROTIME</option>
<option value="APTT" selected>APTT</option>
</select>
<p>Hold down the Ctrl (windows) or Command (Mac) button to select multiple options.</p>
<p><b>Note: If No Selection Made Then All Analytes Are Chosen!</b>.</p>
<br><br>
<label for="analyte_name"><b>Choose an analyte:</b></label>
<br>
<input type="submit" value="Submit">
</form>

<br><br>

</body>

{% endblock %}

```

In []: patient_ranges_tab.html

```

{% extends "parent_templates/employee.html" %}
{% block title %} Patient Ranges Statistical Summary {% endblock %}

{% block main %}
<head>
  <meta charset="UTF-8">
<br>
<h2>Customer Ranges:</h2>
<br>
</head>
<body>

  <br><br>
<h2>Customer Ranges Statistical Summary</h2> <br>
  {{patient_ranges_tab | safe}}

</body>

{% endblock %}

```

In []: place_orders.html

```

{% extends "parent_templates/parent.html" %}
{% block title %}Order Placed{% endblock %}

{% block main %}
<head>
  <meta charset="UTF-8">
  <title>Order Placed</title>
</head>
<body>

<h4>Thank you {{first_name}} {{last_name}} for your order!</h4>
<br><br>
<h4>Your customer id number is: <b>{{id}}<b> </h4>
<br><br>
<h4> Please print or write down this number to gain access to your results and account information.</h4>

</body>
{% endblock %}

```

In []: print_barcode.html

```

{% extends "parent_templates/employee.html" %}
{% block title %}Print Barcode{% endblock %}

{% block main %}
<head>
  <meta charset="UTF-8">
  <title>Print Barcode</title>
</head>
<body>
<br>
<h2>Print Barcode:</h2>
<br>
<p><b>Please enter an order ID number below to display relevant barcodes:</b></p>
<p>You may enter an id any new orders you have created </p>
<p>or</p>
<p>Choose existing order id numbers 100000 through 100900 </p>
<form action="/barcode" method = "POST">
  <p><b>order_id:</b><input type = "text" name = "order_id" /></p>
  <p><input type = "submit" value = "Submit" /></p>
</form>
</body>

{% endblock %}

```

In []: qc_cumulative.html

```

{% extends "parent_templates/employee.html" %}
{% block title %}Quality Control Cumulative Statistics{% endblock %}

```

```
{% block main %}
<body>
<br>
<h2>Quality Control Cumulative Statistics</h2>
<br>
<form action="/qc_cumulative_tab" method = "POST">
  <p><b>Select a start and end date:</b></p>
  <label for="start"><b>Start date:</b></label>
  <input type="date" id="start" name="start_date"
    value="2022-03-01">
  <br><br><br>
  <label for="end"><b>End date:</b></label>
  <input type="date" id="end" name="end_date"
    value="2022-08-08">
  <br><br><br><br>
  <label for="analyte_name"><b>Choose an analyte:</b></label>
  <br>
  <select name="analyte_name" id="analyte_name" >
    <option value="SODIUM" >SODIUM</option>
    <option value="POTASSIUM" >POTASSIUM</option>
    <option value="BICARBONATE" >BICARBONATE</option>
    <option value="CALCIUM" >CALCIUM</option>
    <option value="CHLORIDE" >CHLORIDE</option>
    <option value="GLUCOSE" >GLUCOSE</option>
    <option value="PROTEIN" >PROTEIN</option>
    <option value="CREATININE" >CREATININE</option>
    <option value="GFR" >GFR</option>
    <option value="UREA NITROGEN" >UREA NITROGEN</option>
    <option value="2HRGTT" >2HRGTT </option>
    <option value="A1C" >A1C </option>
    <option value="AST" >AST </option>
    <option value="ALT" >ALT </option>
    <option value="GGT" >GGT </option>
    <option value="ALBUMIN" >ALBUMIN</option>
    <option value="CHOLESTEROL" >CHOLESTEROL</option>
    <option value="TRIGLYCERIDE" >TRIGLYCERIDE</option>
    <option value="LDL" >LDL </option>
    <option value="HDL" >HDL</option>
    <option value="WBC" >WBC </option>
    <option value="RBC" >RBC </option>
    <option value="HEMOGLOBIN" >HEMOGLOBIN</option>
    <option value="HEMATOCRIT" >HEMATOCRIT</option>
    <option value="PLATELET" >PLATELET</option>
    <option value="PROTIME" >PROTIME</option>
    <option value="APTT" >APTT</option>
  </select>
  <br><br>
  <label for="analyte_name"><b>Choose an analyte:</b></label>
  <br>
  <input type="submit" value="Submit">
</form>

<br><br>
</body>

{% endblock %}
```

In []: qc_cumulative_tab.html

```
{% extends "parent_templates/employee.html" %}
{% block title %}Quality Control Cumulative Results{% endblock %}

{% block main %}

<style type="text/css">
ul {
  list-style-type: none;
}
</style>
<head>
  <meta charset="UTF-8">
  <h2>Quality Control Cumulative Results for:</h2>
  <h3>{{analytes[0]}}</h3>
</head>
<body>

  <br><br>
  <b>QC Values Based on Selected Values:</b> <br>
  {{qc_cumulative_tab | safe}}
  <br>

</body>

{% endblock %}
```

In []: qc_scatter.html

```
{% extends "parent_templates/employee.html" %}
{% block title %}Quality Control Cumulative Statistics{% endblock %}

{% block main %}
<body>
<h2>Quality Control Levy Jennings Scatter</h2>

  <form action="/qc_scatter_graph" method = "POST">
    <p><b>Select a start and end date:</b></p>
    <label for="start"><b>Start date:</b></label>
    <input type="date" id="start" name="start_date"
```



```

value="2022-03-01">
<br><br><br>
<label for="end"><b>End date:</b></label>
<input type="date" id="end" name="end_date"
value="2022-08-08">
<br><br><br><br>
<label for="analyte_name"><b>Choose an analyte:</b></label>
<br>
<select name="analyte_name" id="analyte_name" >
<option value="SODIUM" >SODIUM</option>
<option value="POTASSIUM" >POTASSIUM</option>
<option value="BICARBONATE" >BICARBONATE</option>
<option value="CALCIUM" >CALCIUM</option>
<option value="CHLORIDE" >CHLORIDE</option>
<option value="GLUCOSE" >GLUCOSE</option>
<option value="PROTEIN" >PROTEIN</option>
<option value="CREATININE" >CREATININE</option>
<option value="GFR" >GFR</option>
<option value="UREA NITROGEN" >UREA NITROGEN</option>
<option value= "2HRGTT" > 2HRGTT </option>
<option value="A1C" >A1C </option>
<option value="AST" >AST </option>
<option value="ALT" >ALT </option>
<option value="GGT" >GGT </option>
<option value="ALBUMIN" >ALBUMIN</option>
<option value="CHOLESTEROL" >CHOLESTEROL</option>
<option value="TRIGLYCERIDE" >TRIGLYCERIDE</option>
<option value="LDL" >LDL </option>
<option value="HDL" >HDL</option>
<option value="WBC" >WBC </option>
<option value="RBC " >RBC </option>
<option value="HEMOGLOBIN" >HEMOGLOBIN</option>
<option value="HEMATOCRIT" >HEMATOCRIT</option>
<option value="PLATELET" >PLATELET</option>
<option value="PROTIME" >PROTIME</option>
<option value="APTT" >APTT</option>
</select>
<br><br>
<label for="analyte_name"><b>Choose an analyte:</b></label>
<br>
<input type="submit" value="Submit">
</form>

<br><br>
</body>

{% endblock %}

```

In []: qc_summary.html

```

{% extends "parent_templates/employee.html" %}
{% block title %}QC Summary Statistics{% endblock %}

{% block main %}
<head>
<meta charset="UTF-8">
<title>Missing Tests</title>
</head>
<body>
<ul>
<br><br>
<h2>QC Summary Statistics:</h2> <br>
{{qc_summary_display | safe}}
</ul>

</body>

{% endblock %}

```

In []: qc_values.html

```

{% extends "parent_templates/employee.html" %}
{% block title %}Quality Control Individual Results{% endblock %}

{% block main %}
<body>
<br>
<h2>Quality Control Individual Results</h2>
<br>
<form action="/qc_values_tab" method = "POST">
<p><b>Select a start and end date:</b></p>
<label for="start"><b>Start date:</b></label>
<input type="date" id="start" name="start_date"
value="2022-03-01">
<br><br><br>
<label for="end"><b>End date:</b></label>
<input type="date" id="end" name="end_date"
value="2022-03-08">
<br><br><br><br>
<label for="analyte_name"><b>Choose an analyte:</b></label>
<br>
<select name="analyte_name" id="analyte_name" multiple>
<option value="SODIUM" selected>SODIUM</option>
<option value="POTASSIUM" selected>POTASSIUM</option>
<option value="BICARBONATE" selected>BICARBONATE</option>
<option value="CALCIUM" selected>CALCIUM</option>
<option value="CHLORIDE" selected>CHLORIDE</option>
<option value="GLUCOSE" selected>GLUCOSE</option>
<option value="PROTEIN" selected>PROTEIN</option>

```

```

        <option value="CREATININE" selected>CREATININE</option>
        <option value="GFR" selected>GFR</option>
        <option value="UREA NITROGEN" selected>UREA NITROGEN</option>
        <option value="2HRGTT">2HRGTT selected</option>
        <option value="A1C" selected>A1C </option>
        <option value="AST" selected>AST </option>
        <option value="ALT" selected>ALT </option>
        <option value="GGT" selected>GGT </option>
        <option value="ALBUMIN" selected>ALBUMIN</option>
        <option value="CHOLESTEROL" selected>CHOLESTEROL</option>
        <option value="TRIGLYCERIDE" selected>TRIGLYCERIDE</option>
        <option value="LDL" selected>LDL </option>
        <option value="HDL" selected>HDL</option>
        <option value="WBC" selected>WBC </option>
        <option value="RBC selected">RBC </option>
        <option value="HEMOGLOBIN" selected>HEMOGLOBIN</option>
        <option value="HEMATOCRIT" selected>HEMATOCRIT</option>
        <option value="PLATELET" selected>PLATELET</option>
        <option value="PROTIME" selected>PROTIME</option>
        <option value="APTT" selected>APTT</option>
    </select>
    <p>Hold down the Ctrl (windows) or Command (Mac) button to select multiple options.</p>
    <p><b>Note: If No Selection Made Then All Analytes Are Chosen!</b>.</p>
    <br><br>
    <label for="analyte_name"><b>Choose an analyte:</b></label>
    <br>
    <input type="submit" value="Submit">
</form>

<br><br>
</body>

{% endblock %}

```

In []: qc_values_tab.html

```

{% extends "parent_templates/employee.html" %}
{% block title %}Qc_Values{% endblock %}

{% block main %}
<head>
    <meta charset="UTF-8">
    <br>
    <h2>Quality Control Individual Results</h2>
    <br>
</head>
<body>

    <br><br>
    <h2>QC Values Based on Selected Values:</h2> <br>
    {{qc_values_tab | safe}}

</body>

{% endblock %}

```

In []: qc_visualize.html

```

{% extends "parent_templates/employee.html" %}
{% block title %}QC Visualization{% endblock %}

{% block main %}
<body>
<br>
<h2>QC Visualization</h2>
<br>
    <p><b>Select One Analyte to View Levy-Jennings Graphs:</b></p>

    <label for="analyte_name"><b>Choose an analyte:</b></label>
    <br>
<form action="/box_plot" method = "POST">
    <select name="analyte_name" id="analyte_name" >
        <option value="SODIUM">SODIUM</option>
        <option value="POTASSIUM">POTASSIUM</option>
        <option value="BICARBONATE">BICARBONATE</option>
        <option value="CALCIUM">CALCIUM</option>
        <option value="CHLORIDE">CHLORIDE</option>
        <option value="GLUCOSE">GLUCOSE</option>
        <option value="PROTEIN">PROTEIN</option>
        <option value="CREATININE">CREATININE</option>
        <option value="GFR">GFR</option>
        <option value="UREA NITROGEN">UREA NITROGEN</option>
        <option value="2HRGTT">2HRGTT</option>
        <option value="A1C">A1C</option>
        <option value="AST">AST</option>
        <option value="ALT">ALT</option>
        <option value="GGT">GGT</option>
        <option value="ALBUMIN">ALBUMIN</option>
        <option value="CHOLESTEROL">CHOLESTEROL</option>
        <option value="TRIGLYCERIDE">TRIGLYCERIDE</option>
        <option value="LDL">LDL</option>
        <option value="HDL">HDL</option>
        <option value="WBC">WBC</option>
        <option value="RBC">RBC</option>
        <option value="HEMOGLOBIN">HEMOGLOBIN</option>
        <option value="HEMATOCRIT">HEMATOCRIT</option>
        <option value="PLATELET">PLATELET</option>
        <option value="PROTIME">PROTIME</option>
        <option value="APTT">APTT</option>
    </select>

```

```
<br><br>
<label for="analyte_name"><b>Choose an analyte:</b></label>
<br>
<input type="submit" value="Submit">
</form>

<br><br>

</body>

{% endblock %}
```

In []: qc_visualize_graph.html

```
{% extends "parent_templates/employee.html" %}
{% block title %}QC Summary Statistics{% endblock %}

{% block main %}

<style type="text/css">
ul {
  list-style-type: none;
}
</style>

<head>
  <meta charset="UTF-8">
  <title>Levy Jennings Graph for {{analyte_name}}</title>
</head>

<body>
  <img src = '/static/barcode.png'>
  <br><br>

</body>

{% endblock %}
```

In []: results_form.html

```
{% extends "parent_templates/customer.html" %}
{% block title %}Customer Results Dashboard{% endblock %}


{% block main %}
<body>
<h2>Customer Results Dashboard</h2>

  <form action="/results" method = "POST">
    <p><b>Select a start and end date:</b></p>
    <p>Note: start and end date will default to choose all values</p>
    <label for="start"><b>Start date:</b></label>
    <input type="date" id="start" name="start_date"
      value="2022-03-01"
      min="2022-03-01">
    <br><br><br>
    <label for="end"><b>End date:</b></label>
    <input type="date" id="end" name="end_date"
      value="2022-08-08"
      min="2022-03-01">
    <br><br><br><br>
    <select name="alert" id="alert" multiple>
    <option value="Low" selected>Low</option>
    <option value="Normal" selected>Normal</option>
    <option value="High" selected>High</option>
    </select>
    <p>Hold down the Ctrl (windows) or Command (Mac) button to select multiple options.</p>
    <p><b>Note: If No Selection Made Then All Selections Are Chosen!</b></p>
    <br><br><br><br>
    <label for="analyte_name"><b>Choose an analyte:</b></label>
    <br>
    <select name="analyte_name" id="analyte_name" multiple>
    <option value="SODIUM" selected>SODIUM</option>
    <option value="POTASSIUM" selected>POTASSIUM</option>
    <option value="BICARBONATE" selected>BICARBONATE</option>
    <option value="CALCIUM" selected>CALCIUM</option>
    <option value="CHLORIDE" selected>CHLORIDE</option>
    <option value="GLUCOSE" selected>GLUCOSE</option>
    <option value="PROTEIN" selected>PROTEIN</option>
    <option value="CREATININE" selected>CREATININE</option>
    <option value="GFR" selected>GFR</option>
    <option value="UREA NITROGEN" selected>UREA NITROGEN</option>
    <option value="2HRGTT">2HRGTT selected</option>
    <option value="A1C" selected>A1C </option>
    <option value="AST" selected>AST </option>
    <option value="ALT" selected>ALT </option>
    <option value="GGT" selected>GGT </option>
    <option value="ALBUMIN" selected>ALBUMIN</option>
    <option value="CHOLESTEROL" selected>CHOLESTEROL</option>
    <option value="TRIGLYCERIDE" selected>TRIGLYCERIDE</option>
    <option value="LDL" selected>LDL </option>
    <option value="HDL" selected>HDL</option>
    <option value="WBC" selected>WBC </option>
    <option value="RBC selected">RBC </option>
    <option value="HEMOGLOBIN" selected>HEMOGLOBIN</option>
    <option value="HEMATOCRIT" selected>HEMATOCRIT</option>
    <option value="PLATELET" selected>PLATELET</option>
    <option value="PROTIME" selected>PROTIME</option>
    <option value="APTT" selected>APTT</option>
  </select>
  </form>
</body>
</html>
```

```
</select>
  <p>Hold down the Ctrl (windows) or Command (Mac) button to select multiple options.</p>
  <p><b>Note: If No Selection Made Then All Analytes Are Chosen!</b>.</p>
<br><br>
<label for="analyte_name"><b>Choose an analyte:</b></label>
<br>
<input type="submit" value="Submit">
</form>

<br><br>
</body>

{% endblock %}
```


In []:  results_tab.html

```
{% extends "parent_templates/customer.html" %}
{% block title %} Customer Results {% endblock %}

{% block main %}
<head>
  <meta charset="UTF-8">
  <title>Customer Results</title>
</head>
<body>
<br><br>
<h2>Customer Results:</h2> <br>
{{customer_results_display | safe}}

</body>

{% endblock %}
```

In []:  westgard.html

```
{% extends "parent_templates/employee.html" %}
{% block title %}Westgard QC Rules{% endblock %}

{% block main %}
<body>
<br>
<h2>Westgard QC Rules</h2>
<br>
  <form action="/westgard_table" method = "POST">
    <label for="analyte_name"><b>Choose an analyte:</b></label>
    <br>
    <select name="analyte_name" id="analyte_name" >
      <option value="SODIUM" >SODIUM</option>
      <option value="POTASSIUM" >POTASSIUM</option>
      <option value="BICARBONATE" >BICARBONATE</option>
      <option value="CALCIUM" >CALCIUM</option>
      <option value="CHLORIDE" >CHLORIDE</option>
      <option value="GLUCOSE" >GLUCOSE</option>
      <option value="PROTEIN" >PROTEIN</option>
      <option value="CREATININE" >CREATININE</option>
      <option value="GFR" >GFR</option>
      <option value="UREA NITROGEN" >UREA NITROGEN</option>
      <option value= "2HRGTT" > 2HRGTT </option>
      <option value="A1C" >A1C </option>
      <option value="AST" >AST </option>
      <option value="ALT" >ALT </option>
      <option value="GGT" >GGT </option>
      <option value="ALBUMIN" >ALBUMIN</option>
      <option value="CHOLESTEROL" >CHOLESTEROL</option>
      <option value="TRIGLYCERIDE" >TRIGLYCERIDE</option>
      <option value="LDL" >LDL </option>
      <option value="HDL" >HDL</option>
      <option value="WBC" >WBC </option>
      <option value="RBC ">RBC </option>
      <option value="HEMOGLOBIN" >HEMOGLOBIN</option>
      <option value="HEMATOCRIT" >HEMATOCRIT</option>
      <option value="PLATELET" >PLATELET</option>
      <option value="PROTIME" >PROTIME</option>
      <option value="APTT" >APTT</option>
    </select>
    <br><br>
    <label for="analyte_name"><b>Choose an analyte:</b></label>
    <br>
    <input type="submit" value="Submit">
  </form>

<br><br>
</body>

{% endblock %}
```

In []:  westgard_tab.html

```
{% extends "parent_templates/employee.html" %}
{% block title %}Westgard QC Rules{% endblock %}

{% block main %}

<style type="text/css">
ul {
  list-style-type: none;
}
```

```

</style>
<head>
  <meta charset="UTF-8">

  <h2>Westgard Rules :</h2>
  <h3>{{analytes[0]}}</h3>
</head>

<body>
<p>The following rules (Westgard) are designed to alert laboratory staff of more egregious qc results and stop testing:</p>
<p>Standard deviation has been tweaked to help demonstration detection capability of Westgard Rules function</p>

<br>
  {{ display1 | safe }}
<br>

<br>
  {{ display2 | safe }}
<br>

</body>

{% endblock %}

```

in parent template folder (lives inside template folder)

In []: `customer.html`

```

<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.2.1/dist/css/bootstrap.min.css" integrity="sha384-
GJzZqFGwb1QTTN6wy59ffF1BuGJpL5a9DkKMP0DgiMDm4iYMj70gZWKYbI706tWS" crossorigin="anonymous">

    <title>{% block title %}{{ endblock %}}</title>
  </head>
<body>
  <div class="container">
    <div class="row">
      <div class="col-lg-16 text-left">

        <nav class="navbar navbar-expand-lg navbar-light bg-light">
          <a class="navbar-brand" href="#"><img src='/static/beaker.png' width="60" height="60" class="d-inline-block align-top" alt="">
            <h2>Fake-Lab</h2>
          </a>

          <ul class="nav navbar-nav">
            <li class="dropdown">
              <a class="dropdown-toggle"
                data-toggle="dropdown" href="#">
                Contact Info <span class="caret"></span></a>
              <ul class="dropdown-menu">
                <li><a href="/contact_info"> My Profile</a></li>
              </ul>
            </li>

            <li class="dropdown">
              <a class="dropdown-toggle"
                data-toggle="dropdown" href="#">
                Results <span class="caret"></span></a>
              <ul class="dropdown-menu">
                <li><a href="/results_form"> My Results</a></li>
              </ul>
            </li>

            <li class="dropdown">
              <a class="dropdown-toggle"
                data-toggle="dropdown" href="#">
                Billing<span class="caret"></span></a>
              <ul class="dropdown-menu">
                <li><a href="/charges">View My Charges</a></li>
              </ul>
            </li>

            <li class="dropdown">
              <a class="dropdown-toggle"
                data-toggle="dropdown" href="#">
                Return<span class="caret"></span></a>
              <ul class="dropdown-menu">
                <li><a href="/customer_dashboard">Return to Customer Dashboard</a></li>
              </ul>
            </li>
          </ul>
        </nav>


        <main>
          {% block main %}{{ endblock %}}
        </main>

      </div>
    </div>
  </div>
  <!-- Optional JavaScript -->
  <!-- jQuery first, then Popper.js, then Bootstrap JS -->

```

```
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965Dz00rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js" integrity="sha384-wHAiFfRlMFy6i5SRaxvfOCifBUQy1xHdJ/yoi7FRNXMRBu5WHdZYu1hA6ZObLg" crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.2.1/dist/js/bootstrap.min.js" integrity="sha384-B0UglyR+jN6CkvvICOB2joaf5I4l3gm9GU6Hclog6Ls7i6U/mkkaduKaBhlAXv9k" crossorigin="anonymous"></script>
<!-- Custom Jss-->

</body>
</html>
```

In []:  employee.html

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.2.1/dist/css/bootstrap.min.css" integrity="sha384-GJzQfGwb1QTtN6wy59ffF1BuGJpLSa9DkKMP0DgiMDm4iYMj70gZWkYbI706tWS" crossorigin="anonymous">

    <title>{% block title %}{% endblock %}</title>
  </head>
  <body>
    <div class="container">
      <div class="row">
        <div class="col-lg-16 text-left">

          <nav class="navbar navbar-expand-lg navbar-light bg-light">
            <a class="navbar-brand" href="#"><img src='/static/beaker.png' width="60" height="60" class="d-inline-block align-top" alt="">
              <h2>Fake-Lab</h2>
            </a>

            <ul class="nav navbar-nav">
              <li class="dropdown">
                <a class="dropdown-toggle" data-toggle="dropdown" href="#">
                  Logout/Home<span class="caret"></span></a>
                <ul class="dropdown-menu">
                  <li><a href="/employee_login">Logout</a></li>
                  <li><a href="/">Fake-Lab Home Page</a></li>
                </ul>
              </li>

              <ul class="nav navbar-nav">
                <li class="dropdown">
                  <a class="dropdown-toggle" data-toggle="dropdown" href="#">
                    Lab Collections<span class="caret"></span></a>
                  <ul class="dropdown-menu">
                    <li><a href="/collection_tubes">Collection Tubes</a></li>
                    <li><a href="/print_barcode">Reprint Barcode</a></li>
                    <li><a href="/missing_tests">Missing Tests</a></li>
                  </ul>
                </li>

                <ul class="nav navbar-nav">
                  <li class="dropdown">
                    <a class="dropdown-toggle" data-toggle="dropdown" href="#">
                      Testing <span class="caret"></span></a>
                    <ul class="dropdown-menu">
                      <li><a href="/customer_results">Customer Results</a></li>
                      <li><a href="/patient_ranges">Customer Ranges Current Summary</a></li>
                      <li><a href="/moving_averages">Customer Moving Averages/Z_scores/P_values</a></li>
                    </ul>
                  </li>

                  <ul class="nav navbar-nav">
                    <li class="dropdown">
                      <a class="dropdown-toggle" data-toggle="dropdown" href="#">
                        QC Results<span class="caret"></span></a>
                      <ul class="dropdown-menu">
                        <li><a href="/qc_values">Selective QC Individual Results</a></li>
                        <li><a href="/qc_cumulative">Selective QC Aggregated Results</a></li>
                        <li><a href="/qc_summary">All QC Summary Mean/SD</a></li>
                      </ul>
                    </li>

                    <ul class="nav navbar-nav">
                      <li class="dropdown">
                        <a class="dropdown-toggle" data-toggle="dropdown" href="#">
                          QC Visuals/Rules<span class="caret"></span></a>
                        <ul class="dropdown-menu">
                          <li><a href="/qc_scatter">QC Scatter Plots</a></li>
                          <li><a href="/westgard">Westgard Rules</a></li>
                        </ul>
                      </li>
                    </ul>
                  </nav>

                  <main>
                    {% block main %}{% endblock %}
                  </main>

                </div>
              </div>
            </div>
          </div>
```

```
<!-- Optional JavaScript -->
<!-- jQuery first, then Popper.js, then Bootstrap JS -->
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-
q8i/X+965Dz00rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js" integrity="sha384-
wHAiFfRlMfY6i5SRaxvFOCifBUQy1xHdJ/yoi7FRNXMRBu5WHdZYu1hA6ZOblgut" crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.2.1/dist/js/bootstrap.min.js" integrity="sha384-
B0UglyR+jN6CkvvICOB2joaf5I4L3gm9GU6Hc1og6Ls7i6U/mkkaduKaBhlAXv9k" crossorigin="anonymous"></script>
<!-- Custom Jss-->

</body>
</html>
```

In []:  parent.html

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.2.1/dist/css/bootstrap.min.css" integrity="sha384-
GJzZqFGwb1QTTN6wy59ffF1BuGJpL9a9DkKMP0DgiMDm4iYMj70gZWKYbI706tWS" crossorigin="anonymous">

    <title>{% block title %}{% endblock %}</title>
  </head>
<body>
  <div class="container">
    <div class="row">
      <div class="col-lg-12 text-left">

        <nav class="navbar navbar-expand-lg navbar-light bg-light">
          <a class="navbar-brand" href="#">
            <h2>Fake-Lab</h2>
          </a>

          <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false"
aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
          </button>

          <div class="collapse navbar-collapse" id="navbarNav">
            <ul class="navbar-nav">
              <li class="nav-item">
                <a class="nav-link" href="/customer_login">Customer Log-In</a>
              </li>
              <li class="nav-item">
                <a class="nav-link" href="/employee_login">Employee Log-In</a>
              </li>
              <li class="nav-item">
                <a class="nav-link" href="/orders">Place an Order</a>
              </li>
              <li class="nav-item">
                <a class="nav-link" href="/about">About This Project</a>
              </li>
              <li class="nav-item">
                <a class="nav-link" href="/">Return Home</a>
              </li>
            </ul>
          </div>
        </nav>

        <main>
          {% block main %}{% endblock %}
        </main>

      </div>
    </div>
  </div>
</body>
<!-- Optional JavaScript -->
<!-- jQuery first, then Popper.js, then Bootstrap JS -->
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-
q8i/X+965Dz00rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js" integrity="sha384-
wHAiFfRlMfY6i5SRaxvFOCifBUQy1xHdJ/yoi7FRNXMRBu5WHdZYu1hA6ZOblgut" crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.2.1/dist/js/bootstrap.min.js" integrity="sha384-
B0UglyR+jN6CkvvICOB2joaf5I4L3gm9GU6Hc1og6Ls7i6U/mkkaduKaBhlAXv9k" crossorigin="anonymous"></script>
<!-- Custom Jss-->

</body>
</html>
```