Project 3 - Development of a Web Application for Electricity Distribution Company Billing System

Databases - Fall 2024

Due Date: 29 Nov, 2024 11:55 PM

1 Introduction and Overview

This project focuses on the development of a **web application for an electricity distribution billing system**. The application is built using a **three-layer architecture**, as illustrated in Figure 1. This architecture consists of the following components:

- **Web Server**: Responsible for handling HTTP requests from a web browser and sending back results in form of HTML documents. The web server used in this project is **Nginx**.
- **Application Server**: Manages business logic and serves as an intermediary between the web server and the database. The application server used in this project is **Uvicorn**, which hosts Python applications developed using the **FastAPI framework**.
- **Database Server**: Stores and manages the application's data. The database server utilized is **Oracle Database**.

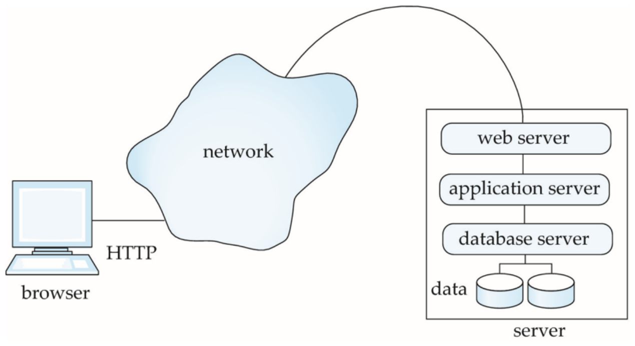

Figure 1: Three-layer web application architecture (source: Database System Concepts, 7th Edition. Silberchatz, Korth, and Sudarshan)

Virtual Machine Setup

In this project, you will create a **virtual machine (VM)** on the **Oracle Cloud Infrastructure (OCI)**. This VM will host both the **web server** and the **application server** for the web application. Detailed instructions and scripts are provided to guide you through the creation of the VM and the configuration of both servers.

Application Development

You are provided with a skeleton code file named `app.py`. This file includes the basic structure for connecting your application to the Oracle Database. You will be using Oracle's library `python-oracledb` to communicate with the database. Find the documentation of the library here: `https://python-oracledb.readthedocs.io/en/latest/index.html`

As part of this project, you will implement the following features:

1. **Database Connection**:
   - Establish a connection between the application and the Oracle Database.
2. **SQL Query Execution**:
   - Write code to send SQL statements from the application to the database.
   - Fetch and display the results of SQL queries in the application.
3. **Stored Procedure Calls**:
   - Execute stored procedures (written in PL/SQL) from the application.

These tasks build upon the work you completed in **Project 1** and **Project 2**, where you implemented SQL queries and stored procedures for the billing system.

2 Requirements

You are required to provide implementations for the following features in the application:

1. Bill retrieval
2. Bill payment processing
3. Bill adjustment processing

2.1 Bill Retrieval

The Bill Retrieval module is responsible for retrieving (querying) an itemized bill based on a customer's usage data for a given month. This section outlines the requirements and expected functionality for the Bill Retreival feature.

2.1.1 Inputs Required

To retrieve (query) a bill, the following input fields must be collected from the user via an HTML form:

- **Customer ID**: A unique identifier for the customer.
- **Connection ID**: A unique identifier for the customer's service connection.
- **Billing Month**: The month for which the bill is being retrieved.
- **Billing Year**: The year for which the bill is being retrieved.

Upon submitting these inputs, the application should retrieve the required bill details, displaying them on a detailed bill page.

2.1.2 Required Bill Details Fields

The following fields must be displayed on the bill details page. These fields should be queried based on the provided input:

- **Customer ID**: The unique identifier of the customer.
- **Connection ID**: The unique identifier of the customer's service connection.
- **Customer Name**: The name of the customer.
- **Customer Address**: The residential or billing address of the customer.
- **Customer Phone**: The contact number of the customer.
- **Customer Email**: The email address of the customer.
- **Connection Type**: Type of connection, such as Residential or Commercial.
- **Division** and **Subdivision**: Division and Subdivision name.
- **Installation Date**: The date on which the connection was installed.
- **Meter Type**: Type of meter installed for this connection.
- **Issue Date**: The date on which the bill was generated.
- **Net Peak Units** and **Net Off-Peak Units**: Total units consumed during peak and off-peak hours for the billing month.
- **Bill Amount**: The total bill amount calculated for the billing period.
- **Due Date** and **Amount After Due Date**: The bill payment due date and the increased amount if paid after the due date.
- **Month**: The billing month for which this bill applies.
- **Year**: The billing year for which this bill applies.
- **Arrears Amount**: Any outstanding amount from previous bills.
- **Fixed Fee Amount**: A fixed fee applied for the billing period.
- **Tax Amount**: The total amount of applicable taxes.
- **Tariffs**: List of applicable tariffs, with each tariff including its name, unit usage, rate, and calculated amount.
- **Taxes**: List of applied taxes with descriptions and amounts.
- **Subsidies**: List and description of subsidies applied to the bill, with their respective amount.
- **Fixed Fees**: List of fixed fees applied to the bill, each with a description and amount.
- **Previous Bills**: A history of the customer's previous bills (up to 10), each including the billing month, year, amount, due date, and payment status.

2.1.3 Output Requirement

Upon successful retrieval, the bill details should be displayed in a structured format on a new bill details page. The output must include all populated fields listed above, accurately reflecting the customer's billing information.

A template HTML file, `bill_retrieval.html` contains the code to display these fields in a structured manner. You can choose to use this, or fork from that, based on your preferences. This template is populated by the application server, which passes a dictionary of values to the template. This dictionary is included in the `app.py` file, and your job will be to populate the dictionary correctly, to get valid outputs displayed on the page.

2.2 Bill Payment Processing

The Bill Payment Processing module is responsible for handling payments made towards a customer's bill. This section outlines the requirements and expected functionality for the Bill Payment Processing feature.

2.2.1 Inputs Required

To process a bill payment, the following input fields must be collected from the user via an HTML form:

- **Bill ID**: The unique identifier for the bill being paid.
- **Amount**: The amount paid by the customer.
- **Payment Method ID**: An identifier for the payment method chosen by the customer (e.g., credit card, bank transfer).

Upon submitting these inputs, the application should process the payment and, if successful, display a payment receipt page.

2.2.2 Required Payment Receipt Fields

The following fields must be displayed on the payment receipt page. These fields should be calculated or queried based on the provided input:

- **Bill ID**: The unique identifier of the bill for which payment was made.
- **Amount**: The amount paid by the customer.
- **Payment Method ID**: The identifier of the payment method used.
- **Payment Method Description**: The description of the payment method used.
- **Payment Date**: The date and time at which the payment was processed.
- **Payment Status**: The status of bill payment as 'Fully Paid' or 'Partially Paid.'
- **Outstanding Amount**: The amount that is outstanding, computed as $billAmount - paidAmount$.

2.2.3 Output Requirement

Upon successful processing, the payment receipt page should display a summary of the payment information. The output must include all populated fields listed above, accurately reflecting the details of the processed payment.

A template HTML file, `bill_payment.html` contains the code to display these fields in a structured manner. You can choose to use this, or fork from that, based on your preferences. This template is populated by the application server, which passes a dictionary of values to the template. This dictionary is included in the `app.py` file, and your job will be to populate the dictionary correctly, to get valid outputs displayed on the page.

2.3 Bill Adjustment Processing

The Bill Adjustment Processing module is responsible for making adjustments to a customer's bill, which may be necessary due to various reasons such as billing errors or applied discounts. This section outlines the requirements and expected functionality for the Bill Adjustment Processing feature.

2.3.1 Inputs Required

To process a bill adjustment, the following input fields must be collected from the user via an HTML form:

- **Bill ID**: The unique identifier for the bill being adjusted.
- **Officer Name**: The name of the officer authorizing the adjustment.
- **Officer Designation**: The designation of the officer authorizing the adjustment.
- **Original Bill Amount**: The original amount of the bill before adjustment.
- **Adjustment Amount**: The amount to adjust (subtracted from the original bill amount).
- **Adjustment Reason**: A brief description of the reason for the adjustment.

Upon submitting these inputs, the application should process the adjustment and, if successful, display an adjustment receipt page.

2.3.2 Required Adjustment Receipt Fields

The following fields must be displayed on the adjustment receipt page. These fields should be calculated or queried based on the provided input:

- **Adjustment ID**: A unique identifier of the adjustment record.
- **Bill ID**: The unique identifier of the bill for which the adjustment was made.
- **Officer Name**: The name of the officer who authorized the adjustment.
- **Officer Designation**: The designation of the officer who authorized the adjustment.
- **Original Bill Amount**: The initial bill amount before any adjustments.
- **Adjustment Amount**: The amount adjusted, which could either increase or decrease the original bill amount.
- **Adjustment Reason**: The reason provided for the adjustment.
- **Adjustment Date**: The date and time at which the adjustment was processed.

2.3.3 Output Requirement

Upon successful processing, the adjustment receipt page should display a summary of the adjustment information. The output must include all populated fields listed above, accurately reflecting the details of the processed adjustment.

For the case of bill adjustments, no template file has been provided. However, feel free to look into the included templates for bill generation and payment processing to get an idea of how to build the template. Looking into HTML will help, although no fancy HTML is required, just a simple structure to display the specified fields.

3 Error Handling

For all the invalid cases e.g. invalid customer ID, or bill ID etc. the application should just return an error alert box. You will be graded on how errors are handled. Note that this response is still coming from the application server. The user should be able to react to the alert box, and stay on the same page. All errors in the specified pages will be dealt with in this way.

4 Getting Started

You have been provided with the following files via LMS:

```
Project3
|-- application
|   |-- app.py
|   |-- requirements.txt
|   |-- static
|   |   `-- style.css
|   `-- templates
|       |-- bill_details.html
|       |-- bill_payment.html
|       |-- bill_retrieval.html
|       |-- index.html
|       `-- payment_receipt.html
|-- env.sh
|-- nginx_setup.sh
|-- oic_setup.sh
`-- python_setup.sh

3 directories, 12 files
```

The application directory contains the source code for the application server. You are allowed to create a directory structure of your own liking, the provided files simply serve as a way to provide a simple reference. The shell scripts included are there to easily set things up for you, on the server.

In order to get started, make sure your Virtual Machine is up and running and you are able to connect to it. You can find details on setting up Virtual Machine on Oracle Cloud in the Section 5

1. If this is the first time setting up, begin by copying the project directory to your server. For example, use the following command:

```
$ scp -i <path-to-ssh-key> -r /path/to/this/directory user@server-ip:/path/to/
    server/directory
```

2. Next, copy your wallet file to the server, placing it inside the `application` directory:

```
$ scp -i <path-to-ssh-key> /path/to/wallet user@server-ip:/path/to/server/
    directory/application
```

3. SSH into your server's VM, and update the path to the wallet file within the `sqlnet.ora` file.

4. Navigate to the project directory and make the setup scripts executable by running:

```
$ chmod +x python_setup.sh oic_setup.sh nginx_setup.sh
```

5. Execute each setup script one by one as follows. Note that these scripts will install and setup the VM to start as a web server. Therefore, **must be run only at the setup**.

```
$ ./python_setup.sh ; ./oic_setup.sh ; ./nginx_setup.sh
```

6. If you receive a warning about ssh connection interruption, that should be fine. You can continue from there.
7. The server will reboot after running these scripts. Wait for 1–2 minutes, log back in, navigate to the project directory, and **make sure to update** `env.sh` with the correct credentials.

4.1 Application deployment

Use the following steps to launch your app, whether locally or remotely.

1. Source the `env.sh` file and activate the virtual environment by running:

```
$ source env.sh ; source venv/bin/activate
```

2. Finally, go to the `application` directory and start the FastAPI server:

```
$ fastapi run app.py
```

If everything was configured correctly, you can test your connection by successfully navigating to `http://<your-server-ip>`.

5 Setting up VM

Your oracle account allows you to create a maximum of 2 VMs, as part of the benefits of your student account. In order to set up the VM, you will need to follow the following steps:

1. Log in to your Oracle cloud account. **Note: If you are one of the students who got locked out of their account, you can contact the course staff for access to the VMs.**
2. You can create a new VM by heading to your "Compute instances" dashboard, or via the hamburger menu -> Compute -> Instances page. Create a new VM.
3. Input relevant details like your VM's name etc.
4. Make sure that you are choosing **Canonical Ubuntu 22.04** as the system image. (not the minimal one). The system shape doesn't need to be changed.
5. Make sure **to save the private key** on your Computer. Don't lose that, as you won't get it later.
6. Make sure that the permissions on the downloaded private key file are correctly configured, i.e. only you can read it. You can find instructions in the official Oracle docs for your relevant platform here to achieve that : (for Windows users, instructions for OpenSSH are relevant) `https://docs.oracle.com/en-us/iaas/Content/GSG/Tasks/testingconnection.htm`.
7. Follow the instructions to then correctly configure the file permissions and to connect to the server. Make sure you can login to your server, over ssh using the following command (get the IP address from the the created VM's dashboard):

```
$ ssh -i /path/to/private/key ubuntu@<ip-address>
```

8. **Important:** Oracle cloud VMs exist in a Virtual Cloud Network (VCN). By default, the VCN is configured to not allow any HTTP traffic. You will need to configure the VCN to allow incoming traffic on port 80 (HTTP). You can do this by navigating to the VCN dashboard, and configuring the security list to allow incoming traffic on this ports.
9. On the VM's dashboard, click on link next to **virtual cloud network**, in the Instance details section.
10. Now click on Security Lists on the left navigation bar for the VCN.
11. Click on the Default Security List.
12. Here you need to open port 80. Add a new ingress rule with the following values:

    Source Type: CIDR

    Source CIDR: 0.0.0.0/0

    IP Protocol: TCP

    Source Port Range: All

    Destination Port Range: 80

    Click on Add Ingress Rules at the bottom.

6 Submission Instructions

You need to submit a zip archive of your modified `application` directory. Make sure to name the archive as follows : `s<your-roll-number>.zip`.

Appendix

Database schema

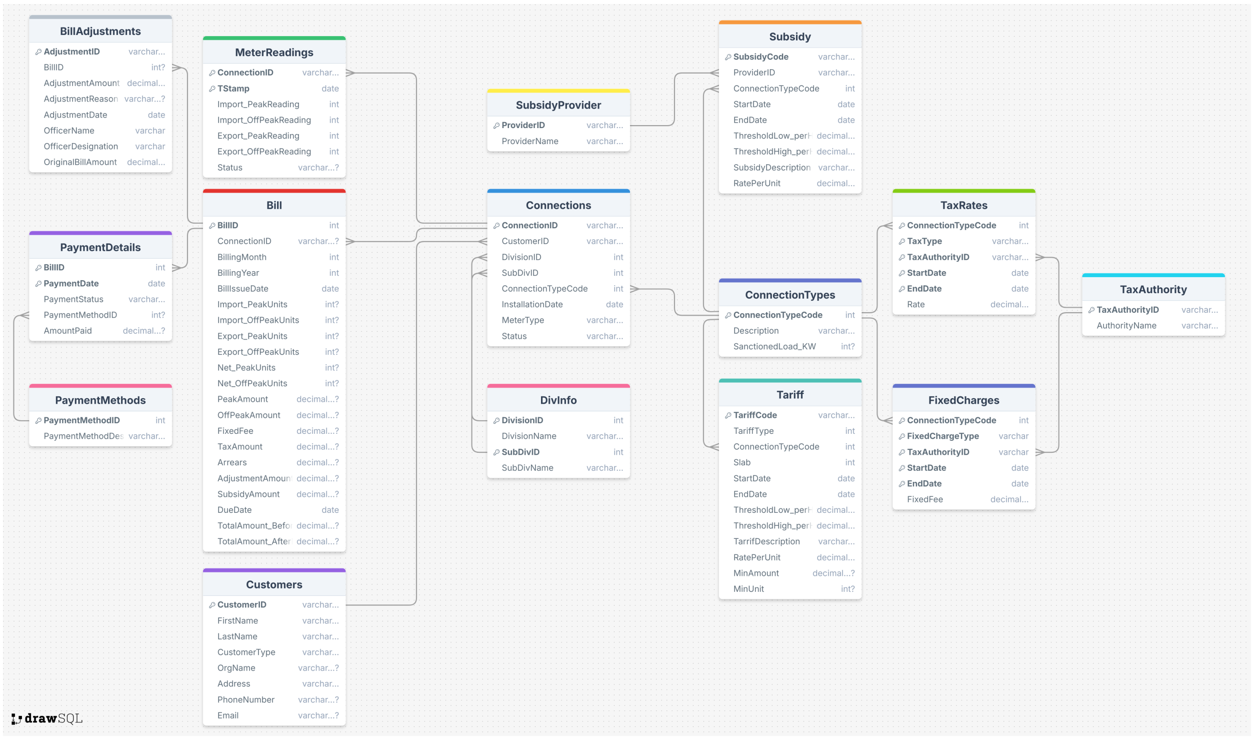

Figure 2: The diagram shows the schema you will be working with. The "key" icon next to attribute names indicates that it is part of the primary key. The corresponding datatype for each attribute is shown here as well. For more details, refer to the file `schema/UtilitySchema.sql`

.