Project 2 - Electricity Distribution Company Billing System

Databases - Fall 2024

Due Date: Monday, November 4 by 11:55 pm

In this project, you will write PL/SQL Stored Procedures and functions to implement the Billing System of an Electricity Distribution Company. The Database Schema of the Billing System is given in the appendix, Figure 1. The SQL files for creating and populating the relevant tables in the database are provided as a zip archive, which can be downloaded from LMS.

Following are the three major tasks of the Billing system for which you will write the PL/SQL Stored Procedures and functions:

1. Bill Generation
2. Bill Payment
3. Bill Adjustments

1 Bill Generation

This task involves generating electricity bills for consumers based on their electricity consumption, applicable tariffs, taxes, and other charges, as well as any governmental subsidies, if applicable. The tariff (i.e., the rate per unit of electricity consumed) may differ across various connection types and is also different for peak and off-peak hours. Tariffs are determined based on the average hourly consumption of electricity. Detailed tariff information is stored in the **Tariff** Table, and meter readings are logged in the **MeterReadings** Table, associated with the respective ConnectionID. Each entry in the **MeterReadings** Table is time-stamped and contains the following data for each ConnectionID:

- `Import_PeakReading`: The number of electricity units consumed during peak hours since the connection start date.
- `Import_OffPeakReading`: The number of electricity units consumed during off-peak hours since the connection start date.
- `Export_PeakReading`: For consumers with solar panels and net metering installed, this field records the number of electricity units produced and sold to the grid during peak hours since the connection start date. For both customers with net metering and conventional meters, this value is ZERO, as the Distribution Company does not purchase electricity from consumers during peak hours.
- `Export_OffPeakReading`: For consumers with solar panels and net metering installed, this field records the number of electricity units produced and sold to the grid during off-peak hours since the connection start date. If the customer does not have solar panels and net metering, this value is ZERO.

In addition to tariffs, various taxes must be applied when generating electricity bills. These taxes may include:

- **Sales Tax**: A percentage of the total bill amount, typically mandated by the government.

Copyright (c) 2024, CS340 - Databases, Lahore University of Management Sciences (LUMS). All rights reserved.

- **Energy Tax**: An additional charge based on the consumer's electricity consumption, which could be applied differently for peak and off-peak usage.
- **Environmental Tax**: This may be levied to promote green energy or discourage excessive consumption of electricity.

Each consumer's total bill will also reflect any applicable subsidies provided by the government, such as reduced rates for low-income households or discounts for energy-efficient practices.

The taxes and subsidies are calculated based on the final electricity consumption values and tariffs. These amounts must be accurately computed and reflected in the total bill.

1.1 Tarrif Amount Computation

The electricity bill for a customer's connection is calculated based on their monthly consumption during both peak and off-peak hours. To prevent overbilling, the applicable tariff is determined based on the average hourly electricity consumption, which is calculated separately for peak and off-peak hours. Below, we explain how the average hourly electricity consumption for the billing month is computed.

For a given connection, let:

- **AHPC** denote hourly electricity consumption during peak hours.
- **AHOC** denote hourly electricity consumption during off-peak hours.
- $\mathbf{Prev_{ImportPeakReading}}$ denote the last `Import_PeakReading` of previous month.
- $\mathbf{Curr_{ImportPeakReading}}$ denote the last `Import_PeakReading` of current month.
- $\mathbf{Prev_{ImportOffPeakReading}}$ denote the last `Import_OffPeakReading` of previous month.
- $\mathbf{Curr_{ImportOffPeakReading}}$ denote the last `Import_OffPeakReading` of current month for the given connection.
- $\mathbf{Prev_{ExportOffPeakReading}}$ denote the last `Export_PeakReading` of previous month.
- $\mathbf{Curr_{ExportOffPeakReading}}$ denote the last `Export_OffPeakReading` of current month for the given connection.
- $\mathbf{Prev_{ExportPeakReading}} = \mathbf{Curr_{ExportPeakReading}} = 0$.

Then,

$$\begin{aligned}\text{BillingDays} = &\ (\text{Timestamp of last reading of current month}) \\ &- (\text{Timestamp of last reading of previous month})\end{aligned} \tag{1}$$

$$\text{PeakUnits}_{\text{Imported}} = (\text{Curr}_{\text{ImportPeakReading}} - \text{Prev}_{\text{ImportPeakReading}}) \tag{2}$$

$$\text{OffPeakUnits}_{\text{Imported}} = (\text{Curr}_{\text{ImportOffPeakReading}} - \text{Prev}_{\text{ImportOffPeakReading}}) \tag{3}$$

$$\text{OffPeakUnits}_{\text{Exported}} = (\text{Curr}_{\text{ExportOffPeakReading}} - \text{Prev}_{\text{ExportOffPeakReading}}) \tag{4}$$

$$\text{AHPC} = \frac{\text{PeakUnits}_{\text{Imported}}}{\text{BillingDays} \times 24} \tag{5}$$

$$\text{AHPC} = \frac{\text{OffPeakUnits}_{\text{Imported}} - \text{OffPeakUnits}_{\text{Exported}}}{\text{BillingDays} \times 24} \tag{6}$$

Once AHPC and AHOC are computed, we need to find the applicable tariff from the Tariff Table. The attributes of Tariff Table are listed below:

- `TariffCode` - Primary key of the Tariff Table

Copyright (c) 2024, CS340 - Databases, Lahore University of Management Sciences (LUMS). All rights reserved.

- `TariffType` - indicate whether the tariff is for peak hour (value = 1) or off-peak hour (value = 2)
- `ConnectionTypeCode`
- `Slab` - Slab number. Tariff slabs are defined for non overlapping ranges of monthly electricity consumption based on the average hourly electricity consumption
- `StartDate` - Date when the tariff becomes effective
- `EndDate` - Date after which the tariff expires
- `ThresholdLow_perHour` - Lower bound on the average hourly electricity consumption for the tariff to become applicable
- `ThresholdHigh_perHour` - Upper bound on the average hourly electricity consumption for the tariff to become applicable
- `TarrifDescription` - Description of the tariff
- `MinUnit` - Minimum number of units for a 30 day period for the corresponding tariff slab. If the customer has consumed less than MinUits in the 30 day persio than the corresponding Tariff slab does not apply. If a customer has consumed more than MinUnits in a 30 day billing period and has hourly consumption within the slab range, then the additional units (Consumed units - MinUnits) are charged at RatePerUnit
- `RatePerUnit` - If a customer has consumed more than MinUnits in a 30 day billing period and has hourly consumption within the slab range, then the additional units (Consumed units - MinUnits) are charged at RatePerUnit
- `MinAmount` - If a customer has consumed more than MinUnits in a 30 day billing period and has hourly consumption within the slab range, then MinAmount is added to the additional amount computed as (additional units * RatePerUnit).

Note that MinUnits and MinAmount are defined for a 30 day billing period. It is possible that the billing period is less than 30 days (e.g., February) or more than 30 days (e.g., month with 31 days). For such billing period the MinUnits and MinAmount are normalized as:

$$\text{NormalizedMinUnits} = \frac{\text{MinUnits} \times \text{ActualDaysInMonth}}{30} \tag{7}$$

$$\text{NormalizedMinAmount} = \frac{\text{MinAmount} \times \text{ActualDaysInMonth}}{30} \tag{8}$$

The applicable tariff tuple is determined by ensuring that the Bill Issue Date is within the interval [`Tariff.StartDate`, `Tariff.EndDate`] and consumer's AHPC is within the open range [`ThresholdLow_perHour`, `ThresholdHigh_perHour`) for peak hour tariff and **AHOC** is within the open range [`ThresholdLow_perHour`, `ThresholdHigh_perHour`) for off-peak hour tariff.

After identifying the applicable tariff tuples, the tariff amounts (PeakAmount and OffPeakAmount) are computed as follows:

Copyright (c) 2024, CS340 - Databases, Lahore University of Management Sciences (LUMS). All rights reserved.

Let t_1, t_2, and t_3 denote the applicable tuples for Import peak tariff, Import off-peak tariff, and export off-peak tariff.

$$\text{AdditionalUnits}_{\text{PeakImport}} = \text{PeakUnits}_{\text{Import}} - \frac{t_1.\text{MinUnits} \times \text{BillingDays}}{30} \tag{9}$$

$$\text{AdditionalUnits}_{\text{OffPeakImport}} = \text{OffPeakUnits}_{\text{Import}} - \frac{t_2.\text{MinUnits} \times \text{BillingDays}}{30} \tag{10}$$

$$\text{AdditionalUnits}_{\text{OffPeakExport}} = \text{OffPeakUnits}_{\text{Export}} - \frac{t_3.\text{MinUnits} \times \text{BillingDays}}{30} \tag{11}$$

$$\begin{aligned}\text{PeakAmount}_{\text{Import}} = {} & (\text{AdditionalUnits}_{\text{PeakImport}} \times t_1.\text{RatePerUnit}) \\ & + \frac{t_1.\text{MinAmount} \times \text{BillingDays}}{30}\end{aligned} \tag{12}$$

$$\begin{aligned}\text{OffPeakAmount}_{\text{Import}} = {} & (\text{AdditionalUnits}_{\text{OffPeakImport}} \times t_2.\text{RatePerUnit}) \\ & + \frac{t_2.\text{MinAmount} \times \text{BillingDays}}{30}\end{aligned} \tag{13}$$

$$\begin{aligned}\text{OffPeakAmount}_{\text{Export}} = {} & (\text{AdditionalUnits}_{\text{OffPeakExport}} \times t_3.\text{RatePerUnit}) \\ & + \frac{t_3.\text{MinAmount} \times \text{BillingDays}}{30}\end{aligned} \tag{14}$$

$$\text{PeakAmount} = \text{PeakAmount}_{\text{Import}} \tag{15}$$

$$\text{OffPeakAmount} = \text{OffPeakAmount}_{\text{Import}} - \text{OffPeakAmount}_{\text{Export}} \tag{16}$$

1.2 Tax Amount Computation

Taxes are applied on the total tariff amount (i.e., PeakAmount + OffPeakAmount). To compute the total tax amount, first all the applicable tuples from TaxRates Table are determined based on the connection type and Bill Issue Date. Suppose the set T denote all the applicable tuples for a given connection in the TaxRates Table. The total Tax amount is computed as:

$$\text{TotalTaxAmount} = \sum_{t \in T} (\text{PeakAmount} + \text{OffPeakAmount}) \times t.\text{Rate} \tag{17}$$

1.3 Fixed Fee Computation

Fixed fees include the charges imposed by the Distribution Company or Government authorities that are added to the bill. To calculate the fixed fee, all applicable tuples from the FixedCharges Table are first identified based on the connection type and Bill Issue Date. Let the set F represent all applicable entries for a given connection. The total FixedFee is then calculated as:

$$\text{TotalFixedFee} = \sum_{f \in F} f.\text{Amount} \tag{18}$$

1.4 Subsidy Amount Computation

Government (Provincial or Federal) may provide subsidy on electricity bill for certain consumers depending on their connection type and electricity consumption. To compute the total subsidy amount, first all the applicable tuples from subsidy Table are identified based on the connection type, Bill Issue Date, and the average hourly electricity consumption in terms of unit per hour computed as follows:

$$\text{UnitPerHour}_{\text{subsidy}} = \frac{\text{PeakUnits}_{\text{Imported}} + \text{OffPeakUnits}_{\text{Imported}}}{\text{BillingDays} \times 24} \tag{19}$$

Suppose the set S denote all the applicable tuples for a given connection in the Subsidy Table. The total Subsidy Amount is computed as:

$$\text{SubsidyAmount} = \sum_{s \in S} (\text{UnitPerHour}_{\text{subsidy}} \times 24 \times \text{BillingDays}) \times s.\text{RatePerUnit} \tag{20}$$

Copyright (c) 2024, CS340 - Databases, Lahore University of Management Sciences (LUMS). All rights reserved.

1.5 Arrears Computation

Arrears refer to the unpaid or overdue amount from last bill. When calculating arrears, the computation involves:

- Unpaid Amount of Last Bill: If the entire bill remains unpaid, the full amount (total amount after due date) of the previous bill becomes arrears.
- Remaining Amount of a Partially Paid Bill: If a bill was partially paid, the outstanding balance from that bill will be carried forward as arrears.

1.6 Total Bill Amount

The total bill amount before due date is computed as:

$$\begin{aligned}\text{TotalAmount}_{\text{BeforeDueDate}} = (&\text{PeakAmount} + \text{OffPeakAmount} + \text{TaxAmount} + \text{FixedFee}) \\ &- (\text{SubsidyAmount} + \text{AdjustmentAmount}) + \text{Arrears}\end{aligned} \tag{21}$$

Note: The AdjustmentAmount is set to ZERO when the bill is first generated. An authorized officer from the Distribution Company may adjust the bill in response to a valid user complaint (e.g., incorrect reading, defective meter, overbilling, etc.), as detailed in Task 3. Once the bill is adjusted, the total amount due is recalculated accordingly.

The bill payment is due within 10 days from the Bill Issue Date. Therefore:

- DueDate = BillIssueDate+10, and
- $\text{TotalAmount}_{\text{AfterDueDate}} = \text{TotalAmount}_{\text{BeforeDueDate}} \times 1.10$

2 Task 1: PL/SQL functions for Bill Generation

You need to write the following PL/SQL functions and procedures for Task 1.

2.1 Functions to Compute Electricity Consumption for Individual Consumers

This Section provides the interface specifications of PL/SQL functions that you will develop to compute monthly electricity consumption metrics of individual consumers for generating their bill. The relevant table for computing these metrics is **MeterReadings** Table.

2.1.1 Function to compute BillingDays

Write a function called, `fun_compute_BillingDays`, that computes the number of BillingDays in given month and year based on meter reading entries in the **MeterReadings** Table. Refer to Equation (1) in Section 1.1 for the method used to compute the billing days. The interface specification for `fun_compute_BillingDays` is provided below:

```
function fun_compute_BillingDays (
    p_ConnectionID  IN  VARCHAR2,
    p_BillingMonth  IN  NUMBER,
    p_BillingYear   IN  NUMBER
    )
RETURN NUMBER;
```

Parameters:

- `p_ConnectionID`: An input parameter that passes the Connection ID of the consumer.

Copyright (c) 2024, CS340 - Databases, Lahore University of Management Sciences (LUMS). All rights reserved.

- `p_BillingMonth`: An input parameter that passes the billing month.
- `p_BillingYear`: An input parameter that passes the billing year.

2.1.2 Function to compute ImportPeakUnits

Write a function called, `fun_compute_ImportPeakUnits`, that computes the number of ImportPeakUnits in a given month and year based on meter reading entries the **MeterReadings** Table. Refer to Equation (2) in Section 1.1 for the method used to compute ImportPeakUnits. The interface specification for `fun_compute_ImportPeakUnits` is provided below:

```
function fun_compute_ImportPeakUnits (
    p_ConnectionID  IN  VARCHAR2,
    p_BillingMonth  IN  NUMBER,
    p_BillingYear   IN  NUMBER
)
RETURN NUMBER;
```

Parameters:

- `p_ConnectionID`: An input parameter that passes the Connection ID of the consumer.
- `p_BillingMonth`: An input parameter that passes the billing month.
- `p_BillingYear`: An input parameter that passes the billing year.

The function returns the ImportPeakUnits for the given month and year against the specified `p_ConnectionID` if executed successfully. In case of an error, the function returns -1.

2.1.3 Function to compute ImportOffPeakUnits

Write a function called, `fun_compute_ImportOffPeakUnits`, that computes the number of ImportOffPeakUnits in a given month and year based on meter reading entries in the **MeterReadings** Table. Refer to Equation (3) in Section 1.1 for the method used to compute ImportOffPeakUnits. The interface specification for `fun_compute_ImportOffPeakUnits` is provided below:

```
function fun_compute_ImportOffPeakUnits (
    p_ConnectionID  IN  VARCHAR2,
    p_BillingMonth  IN  NUMBER,
    p_BillingYear   IN  NUMBER
)
RETURN NUMBER;
```

Parameters:

- `p_ConnectionID`: An input parameter that passes the Connection ID of the consumer.
- `p_BillingMonth`: An input parameter that passes the billing month.
- `p_BillingYear`: An input parameter that passes the billing year.

The function returns the ImportOffPeakUnits for the given month and year against the specified `p_ConnectionID` if executed successfully. In case of an error, the function returns -1.

Copyright (c) 2024, CS340 - Databases, Lahore University of Management Sciences (LUMS). All rights reserved.

2.1.4 Function to compute ExportOffPeakUnits

Write a function called, `fun_compute_ExportOffPeakUnits`, that computes the number of ExportOffPeakUnits in a given month and year based on meter reading entries in the **MeterReadings** Table. Refer to Equation (4) in Section 1.1 for the method used to compute ExportOffPeakUnits. The interface specification for `fun_compute_ExportOffPeakUnits` is provided below:

```
function fun_compute_ExportOffPeakUnits (
    p_ConnectionID  IN  VARCHAR2,
    p_BillingMonth  IN  NUMBER,
    p_BillingYear   IN  NUMBER
)
RETURN NUMBER;
```

Parameters:

- `p_ConnectionID`: An input parameter that passes the Connection ID of the consumer.
- `p_BillingMonth`: An input parameter that passes the billing month.
- `p_BillingYear`: An input parameter that passes the billing year.

The function returns the ExportOffPeakUnits for the given month and year against the specified `p_ConnectionID` if executed successfully. In case of an error, the function returns -1.

2.2 Functions to Compute Metrics for Tariff Amount

This section provides the interface specifications for the PL/SQL functions that you will develop to compute tariff amount metrics for individual consumers' bill generation. The relevant tables used for computing these metrics are **TariffRates**, **Connections**, and **ConnectionTypes**.

2.2.1 Function to compute PeakAmount

Write a function called, `fun_compute_PeakAmount`, that computes the PeakAmount for individual consumers' bill generation. Refer to Equations (9), (12), and (15) in Section 1.1 for the method used to compute the PeakAmount. The interface specification for `fun_compute_PeakAmount` is provided below:

```
function fun_compute_PeakAmount (
    p_ConnectionID  IN  VARCHAR2,
    p_BillingMonth  IN  NUMBER,
    p_BillingYear   IN  NUMBER,
    p_BillIssueDate IN  DATE
)
RETURN NUMBER;
```

Parameters:

- `p_ConnectionID`: An input parameter that passes the Connection ID of the consumer.
- `p_BillingMonth`: An input parameter that passes the billing month.
- `p_BillingYear`: An input parameter that passes the billing year.
- `p_BillIssueDate`: An input parameter that passes the Bill Issue Date for determining the applicable tariff(s).

The function returns the PeakAmount (rounded to two decimal places) for the given month and year against the specified `p_ConnectionID` if executed successfully. In case of an error, the function returns -1.

Copyright (c) 2024, CS340 - Databases, Lahore University of Management Sciences (LUMS). All rights reserved.

2.2.2 Function to compute OffPeakAmount

Write a function called, `fun_compute_OffPeakAmount`, that computes the OffPeakAmount for individual consumers' bill generation. Refer to Equations (10), (11), (13), (14) and (16) in Section 1.1 for the method used to compute the OffPeakAmount. The interface specification for `fun_compute_OffPeakAmount` is provided below:

```
function fun_compute_OffPeakAmount (
    p_ConnectionID  IN  VARCHAR2,
    p_BillingMonth  IN  NUMBER,
    p_BillingYear   IN  NUMBER,
    p_BillIssueDate IN  DATE
)
RETURN NUMBER;
```

Parameters:

- `p_ConnectionID`: An input parameter that passes the Connection ID of the consumer.
- `p_BillingMonth`: An input parameter that passes the billing month.
- `p_BillingYear`: An input parameter that passes the billing year.
- `p_BillIssueDate`: An input parameter that passes the Bill Issue Date for determining the applicable tariff(s).

The function returns the OffPeakAmount (rounded to two decimal places) for the given month and year against the specified `p_ConnectionID` if executed successfully. In case of an error, the function returns -1.

2.3 Functions to Compute Tax, Arrears, and Subsidy for Total Bill

This section provides the interface specifications for the PL/SQL functions that you will develop to compute the various billing metrics, including the tax, arrears, and subsidy for individual consumers during bill generation. The relevant tables used for computing these metrics include TaxRates, **FixedCharges**, **Subsidy**, **Connections**, and **ConnectionTypes**.

2.3.1 Function to compute TaxAmount

Write a function called, `fun_compute_TaxAmount`, that computes the total TaxAmount for individual consumers' bill generation. Refer to Equation (17) in Section 1.2 for the method used to compute the TaxAmount. The interface specification for `fun_compute_TaxAmount` is provided below:

```
function fun_compute_TaxAmount (
    p_ConnectionID   IN  VARCHAR2,
    p_BillingMonth   IN  NUMBER,
    p_BillingYear    IN  NUMBER,
    p_BillIssueDate  IN  DATE,
    p_PeakAmount     IN  NUMBER,
    p_OffPeakAmount  IN  NUMBER
)
RETURN NUMBER;
```

Parameters:

- `p_ConnectionID`: An input parameter that passes the Connection ID of the consumer.
- `p_BillingMonth`: An input parameter that passes the billing month.

Copyright (c) 2024, CS340 - Databases, Lahore University of Management Sciences (LUMS). All rights reserved.

- `p_BillingYear`: An input parameter that passes the billing year.
- `p_BillIssueDate`: An input parameter that passes the Bill Issue Date for determining the applicable tax(es).
- `p_PeakAmount`: An input parameter that passes the Peak Amount for tax computation based on applicable tax rate(s).
- `p_OffPeakAmount`: An input parameter that passes the OffPeak Amount for tax computation based on applicable tax rate(s).

The function returns the TaxAmount (rounded to two decimal places) for the given month and year against the specified `p_ConnectionID` if executed successfully. In case of an error, the function returns -1.

2.3.2 Function to compute FixedFee Amount

Write a function called, `fun_compute_FixedFee`, that computes the FixedFee for individual consumers' bill generation. Refer to Equation (18) in Section 1.3 for the method used to compute the FixedFee. The interface specification for `fun_compute_FixedFee` is provided below:

```
function fun_compute_FixedFee (
    p_ConnectionID   IN  VARCHAR2,
    p_BillingMonth   IN  NUMBER,
    p_BillingYear    IN  NUMBER,
    p_BillIssueDate  IN  DATE
)
RETURN NUMBER;
```

Parameters:

- `p_ConnectionID`: An input parameter that passes the Connection ID of the consumer.
- `p_BillingMonth`: An input parameter that passes the billing month.
- `p_BillingYear`: An input parameter that passes the billing year.
- `p_BillIssueDate`: An input parameter that passes the Bill Issue Date for determining the applicable Fixed Fee tuple(s).

The function returns the FixedFee (rounded to two decimal places) for the given month and year against the specified `p_ConnectionID` if executed successfully. In case of an error, the function returns -1.

2.3.3 Function to compute Arrears

Write a function called, `fun_compute_Arrears`, that computes the total Arrears for individual consumers' bill generation. Refer to Section 1.5 for the method used to compute the arrears. The interface specification for `fun_compute_Arrears` is provided below:

```
function fun_compute_Arrears (
    p_ConnectionID   IN  VARCHAR2,
    p_BillingMonth   IN  NUMBER,
    p_BillingYear    IN  NUMBER,
    p_BillIssueDate  IN  DATE
)
RETURN NUMBER;
```

Parameters:

Copyright (c) 2024, CS340 - Databases, Lahore University of Management Sciences (LUMS). All rights reserved.

- `p_ConnectionID`: An input parameter that passes the Connection ID of the consumer.
- `p_BillingMonth`: An input parameter that passes the billing month.
- `p_BillingYear`: An input parameter that passes the billing year.
- `p_BillIssueDate`: An input parameter that passes the Bill Issue Date for determining the arrears.

The function returns the Arrears (rounded to two decimal places) for the given month and year against the specified `p_ConnectionID` if executed successfully. In case of an error, the function returns -1.

2.3.4 Function to compute SubsidyAmount

Write a function called, `fun_compute_SubsidyAmount`, that computes the total SubsidyAmount for individual consumers' bill generation. Refer to Equations (19) and (20) in Section 1.4 for the method used to compute the SubsidyAmount. The interface specification for `fun_compute_SubsidyAmount` is provided below:

```
function fun_compute_SubsidyAmount (
    p_ConnectionID        IN  VARCHAR2,
    p_BillingMonth        IN  NUMBER,
    p_BillingYear         IN  NUMBER,
    p_BillIssueDate       IN  DATE,
    p_ImportPeakUnits     IN  NUMBER,
    p_ImportOffPeakUnits  IN  NUMBER
)
RETURN NUMBER;
```

Parameters:

- `p_ConnectionID`: An input parameter that passes the Connection ID of the consumer.
- `p_BillingMonth`: An input parameter that passes the billing month.
- `p_BillingYear`: An input parameter that passes the billing year.
- `p_BillIssueDate`: An input parameter that passes the Bill Issue Date for determining the applicable subsidy rate(s).
- `p_ImportPeakUnits`: An input parameter that passes the Import Peak Units for subsidy computation based on applicable subsidy rate(s).
- `p_ImportOffPeakUnits`: An input parameter that passes the Import Off-Peak Units for subsidy computation based on applicable subsidy rate(s).

The function returns the SubsidyAmount (rounded to two decimal places) for the given month and year against the specified `p_ConnectionID` if executed successfully. In case of an error, the function returns -1.

2.4 Functions to Generate Bill

This section provides the interface specifications for the PL/SQL functions that you will develop to generate electricity bills for individual consumers.

Copyright (c) 2024, CS340 - Databases, Lahore University of Management Sciences (LUMS). All rights reserved.

2.4.1 Function to generate Bill by inserting records in the Bill Table

Write a function called, `fun_generate_Bill`, that generates a bill for a consumer by inserting a record (tuple) into the **Bill Table** for a specified month and year. This function may call one or more of the previously defined functions to compute values for the respective attributes in the Bill Table. Refer to Section 1.6 for details on the DueDate, TotalAmount_BeforeDueDate, TotalAmount_AfterDueDate, and AdjustmentAmount. The interface specification for `fun_generate_Bill` is provided below:

```
function fun_Generate_Bill (
    p_BillID        IN  NUMBER,
    p_ConnectionID  IN  VARCHAR2,
    p_BillingMonth  IN  NUMBER,
    p_BillingYear   IN  NUMBER,
    p_BillIssueDate IN  DATE
)
RETURN NUMBER;
```

Parameters:

- `p_BillID`: A unique identifier for the bill (input parameter).
- `p_ConnectionID`: The consumer's connection ID (input parameter).
- `p_BillingMonth`: The month for which the bill is being generated (input parameter).
- `p_BillingYear`: The year for which the bill is being generated (input parameter).
- `p_BillIssueDate`: The date the bill is issued (input parameter).

The function returns 1 if executed successfully, and -1 if it encounters an error..

2.4.2 Function for generating monthly bills of all consumers

Write a function called, `fun_batch_Billing`, that generates bills for all consumers for a specified month and year. This function calls the function `fun_generate_Bill` in a loop for each 'Active' connection for the given month and year, assigning a unique `BillID` for each. The interface specification for `fun_batch_Billing` is provided below:

```
function fun_batch_Billing (
    p_BillingMonth  IN  NUMBER,
    p_BillingYear   IN  NUMBER,
    p_BillIssueDate IN  DATE
)
RETURN NUMBER;
```

Parameters:

- `p_BillingMonth`: The month for which the bill is being generated (input parameter).
- `p_BillingYear`: The year for which the bill is being generated (input parameter).
- `p_BillIssueDate`: The date the bill is issued (input parameter).

The function returns the total number of bills successfully generated for the specified month and year. If no bills are inserted due to errors or other issues, the function returns -1.

Copyright (c) 2024, CS340 - Databases, Lahore University of Management Sciences (LUMS). All rights reserved.

3 Task 2: Bill Payment

This task involves processing bill payments made by consumers against their issued electricity bills. Once a bill is generated and issued, consumers can make payments either in full or in part. The PaymentDetails Table stores information related to each payment transaction, linked to the respective `BillID`.

Each entry in the **PaymentDetails** Table records the following data for each payment:

- `BillID`: The identifier of the bill for which the payment is being made.
- `PaymentDate`: The date the payment was made.
- `PaymentStatus`: The status of payment, which would be one of the two values 'FULLY PAID' or Partially PAID'.
- `PaymentMethodID`: The ID of Payment method used (e.g., credit card, bank transfer, cash). The applicable payment methods are defined in PaymentMethods Table.
- `PaymentAmount`: The total amount paid by the consumer in the transaction.
- `PaymentID`: Primary key of the PaymentDetails Table.

If a consumer makes a partial payment, the remaining balance is carried over to the next billing cycle as arrears (see Section 1.5).

3.1 Interface Specification for Payment Processing

This section provides the interface specifications for the PL/SQL function that you will develop to process bill payment by consumers.

3.1.1 Function to process and record payment

Write a function called, `fun_process_Payment`, that records payment made by a consumer for a given `BILLID`. The payment can be paid in full or in part as explained above. The interface specification for `fun_process_Payment` is provided below:

```
function fun_process_Payment (
    p_BILLID          IN  NUMBER,
    p_PaymentDate     IN  DATE,
    p_PaymentMethodID IN  NUMBER,
    p_AmountPaid      IN  NUMBER
)
RETURN NUMBER;
```

Parameters:

- `p_BILLID`: This is the unique identifier for the bill being paid. It links the payment to a specific bill issued to the consumer (input parameter).
- `p_PaymentDate`: The date on which the payment is made by the consumer (input parameter). This value is important for determining any late fees on the payment date relative to the bill's due date.
- `p_PaymentMethodID`: This parameter identifies the method used for the payment (input parameter). Each payment method, such as credit card, bank transfer, or cash, is associated with a unique identifier stored in the system.
- `p_AmountPaid`: This is the total amount paid by the consumer during the transaction (input parameter). It can be either the full amount due on the bill or a partial payment. In case of partial payment, the remaining balance will be carried forward.

Copyright (c) 2024, CS340 - Databases, Lahore University of Management Sciences (LUMS). All rights reserved.

The function returns '1' if the payment is successfully processed and recorded. It returns '-1' if the payment fails due to an error, such as an invalid bill ID, payment method, or other system issues.

4 Task 3: Bill Adjustment

This task involves making adjustments to a consumer's bill when necessary. An authorized officer from the Distribution Company can adjust the bill in response to a valid user complaint, such as:

- **Incorrect Reading**: If the meter reading was inaccurately recorded.
- **Defective Meter**: When a faulty meter leads to inaccurate billing.
- **Overbilling**: If the consumer is charged for more than their actual consumption.

Once the authorized adjustment is made, the total amount due on the bill is recalculated to reflect the correction. The adjustment amount is also recorded in the Bill Table. For recalculation of the total bill amount refer to Section 1.6.

To ensure transparency, every bill adjustment is recorded with details, including the date of adjustment, name and designation of the officer making the adjustment, original bill amount, adjustment amount and the reason for the adjustment. This information is stored in the BillAdjustment Table and linked to the corresponding BillID for audit purposes. In the event of a dispute or inquiry, this record provides a clear trail of all modifications made to the bill.

4.1 Interface Specification for Bill Adjustment

This section provides the interface specifications for the PL/SQL function that you will develop to make adjustment to a consumer bill.

4.1.1 Function to make Bill adjustment

Write a function called, `fun_adjust_Bill`, that records a bill adjustment made by an authorized officer of the Distribution Company for a given `BILLID`. The interface specification for `fun_adjust_Bill` is provided below:

```
function fun_adjust_Bill (
    p_AdjustmentID    IN  NUMBER,
    p_BILLID          IN  NUMBER,
    p_AdjustmentDate  IN  DATE,
    p_OfficerName     IN  VARCHAR2,
    p_OfficerDesignation     IN  VARCHAR2,
    p_OriginalBillAmount     IN  NUMBER,
    p_AdjustmentAmount       IN  NUMBER,
    p_AdjustmentReason       IN  VARCHAR2
)
RETURN NUMBER;
```

Parameters:

- `p_AdjustmentID`: A unique identifier for the adjustment (input parameter). This ID ensures that each adjustment is logged with a distinct record for traceability and audit purposes.
- `p_BILLID`: The identifier of the bill being adjusted (input parameter). This links the adjustment to a specific bill issued to a consumer, ensuring that the correct bill is modified.

Copyright (c) 2024, CS340 - Databases, Lahore University of Management Sciences (LUMS). All rights reserved.

- `p_AdjustmentDate`: The date on which the adjustment is made (input parameter). This date is important for record-keeping and auditing, and helps in calculating any late fees or discrepancies arising due to timing.
- `p_OfficerName`: The name of the authorized officer making the adjustment (input parameter). This field provides accountability, recording which officer was responsible for approving and applying the adjustment.
- `p_OfficerDesignation`: The designation of the officer making the adjustment (input parameter). This provides additional context about the authority level of the officer, which may be important in audit trails and future inquiries.
- `p_OriginalBillAmount`: The original total amount of the bill before the adjustment (input parameter). This value is necessary for recalculating the new bill total and for documenting the adjustment history.
- `p_AdjustmentAmount`: The amount by which the bill is being adjusted (input parameter). The system uses this amount to recalculate the total amount due.
- `p_AdjustmentReason`: The reason for the adjustment (input parameter). This field records why the adjustment was made, such as incorrect meter reading, overbilling, or faulty meter. This reason is crucial for transparency and for resolving disputes or inquiries regarding the adjustment.

The function returns '1' if the bill adjustment is successfully recorded. If the adjustment fails due to an error (e.g., invalid `BILLID` or adjustment details), the function returns '-1'.

5 Getting Started

1. Go to the directory where you have the `.zip` file that you downloaded from LMS.
2. Extract the contents of the archive.
3. Open the directory, which contains the extracted contents of the archive, in the terminal of your choice (`Powershell/Bash/Zsh`). For example:

```
$ cd path/to/project1
$ ls
dropall.sql  myexec.sql  populate  schema  submission
```

4. Once the database is up and running, connect to your database instance, using the `sqlplus` client, by running the following command (as done in Project 0).

```
$ sqlplus <user_name>/<your_password>@<connection_alias>
```

5. Once you are connected to the database, you can run the `myexec.sql` script, by running the following command (inside `sqlplus`):

```
> @myexec;
```

   This will drop the previously created tables, create the new tables, and populate them with data. The process should take around 20-30 minutes. Wait for the database to get populated correctly with all the values.

6. You are provided with the file called `functions.sql`. The file contains the skeleton of the functions that you need to implement. You need to fill in the function bodies with the required logic. Make sure to not change the function names or the parameters.

Copyright (c) 2024, CS340 - Databases, Lahore University of Management Sciences (LUMS). All rights reserved.

7. In order to compile your functions, you can simply run the following command in the `sqlplus` client:

```
> @functions;
```

8. Don't worry if you get errors, you can see the compilation errors by running the following:

```
> show errors;
```

9. The file `test_functions.sql` contains calls to the functions that you have implemented. You can run this file to test your functions. The file is also supposed to show you how to set up your own tests, by calling these functions.

6 Submission Instructions

You only need to submit a single `.sql` file, containing all of your function definitions. Use the file `functions.sql` for this purpose. Make sure to rename the file as `s<your_roll_number>_functions.sql`, e.g. `s24100173_functions.sql`.

Copyright (c) 2024, CS340 - Databases, Lahore University of Management Sciences (LUMS). All rights reserved.

Appendix

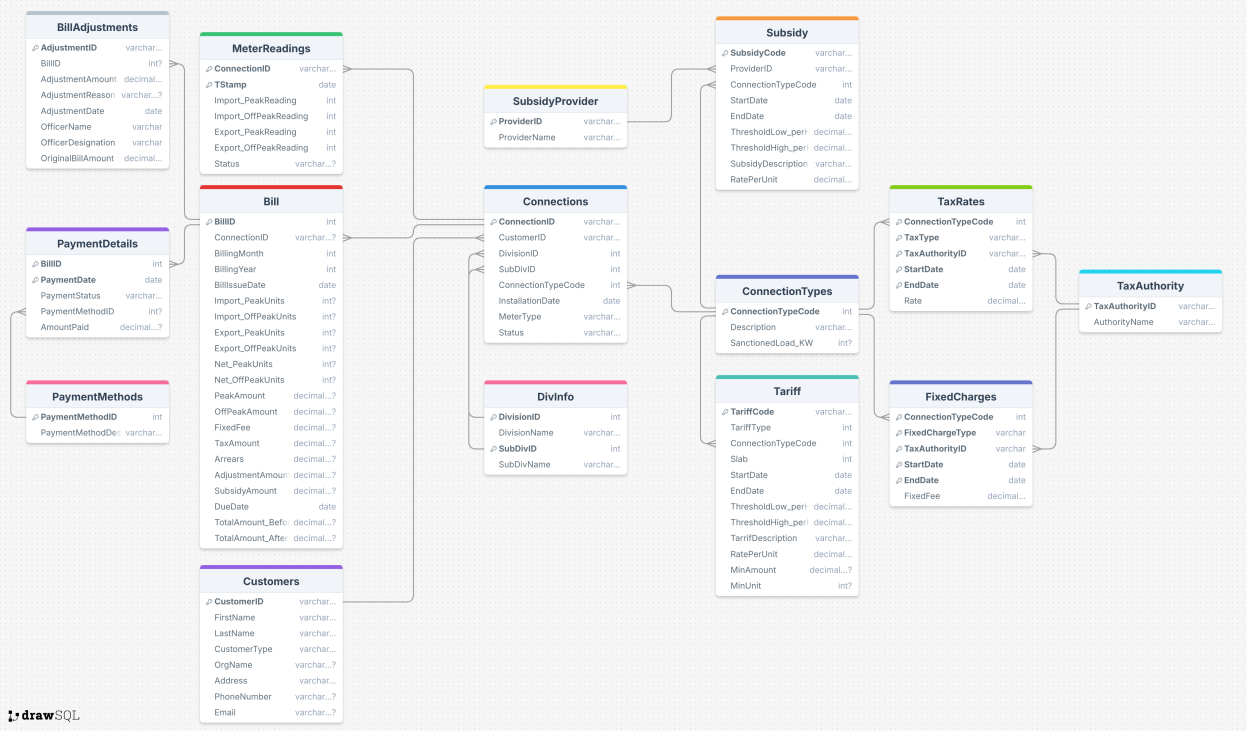

Figure 1: Database Schema

Copyright (c) 2024, CS340 - Databases, Lahore University of Management Sciences (LUMS). All rights reserved.