Carbon Certificate Management System

By

Dhruvika Rajput

ITM SLS Baroda University

Bachelors in Engineering (CSE)

SAP Educate to Employ (E2E)

Software Development Track

Index

RESTFUL APPLICATION

5. **Project Description**
6. **Database Schema (Persistence Layer)**
   6.1. ZEM_SOURCES – Source Table
   6.2. ZEM_FACTOR_TABLE – Factor Table
   6.3. ZEM_USAGE_LOG– Usage Log Table
   6.4. ZEM_CERTIFICATE – Certificate Issue Table
7. **CDS Views (Data Model Layer)**
   7.1. Data Definitions
   7.1.1. *ZEM_USAGE_ENTITY*
   7.1.2. *ZEM_EMISSION*
   7.1.3. *ZI_CREDIT_REPORT*
   7.1.4. *ZI_ELIGCERTIFICATE*
   7.2. *Metadata Extensions (UI)*
   7.2.1. ZC_EM_USAGE_LOG000
   7.2.2. ZI_EMISSIONREPORT_UI
   7.2.3. ZI_CERTIFICATE_UI
8. **Service Definition & Binding**
   8.1. Service Definitions
   8.2. Service Bindings
   8.2.1. USAGE_LOG_TABLE
   8.2.2. EMISSION REPORT
   8.2.3. CERTIFICATE SUMMARY DASHBOARD
9. **Behaviour Implementation Classes**
   9.1. ZBP_CFM_USAGE_ENTITY
   9.2. ZBP_CERTIFY
10. **Final Outcome**
11. **Conclusion**

1. Project Description

Overview

Carbon Certificate Management System is a SAP ABAP-based backend solution designed for enterprises to manage, monitor, and certify carbon emissions. It captures energy usage, calculates emissions using defined emission factors, assigns carbon credits, and generates official carbon certificates with automated expiry and duplicate checks. The system supports sustainability tracking and environmental compliance in alignment with ESG goals.

Key Features

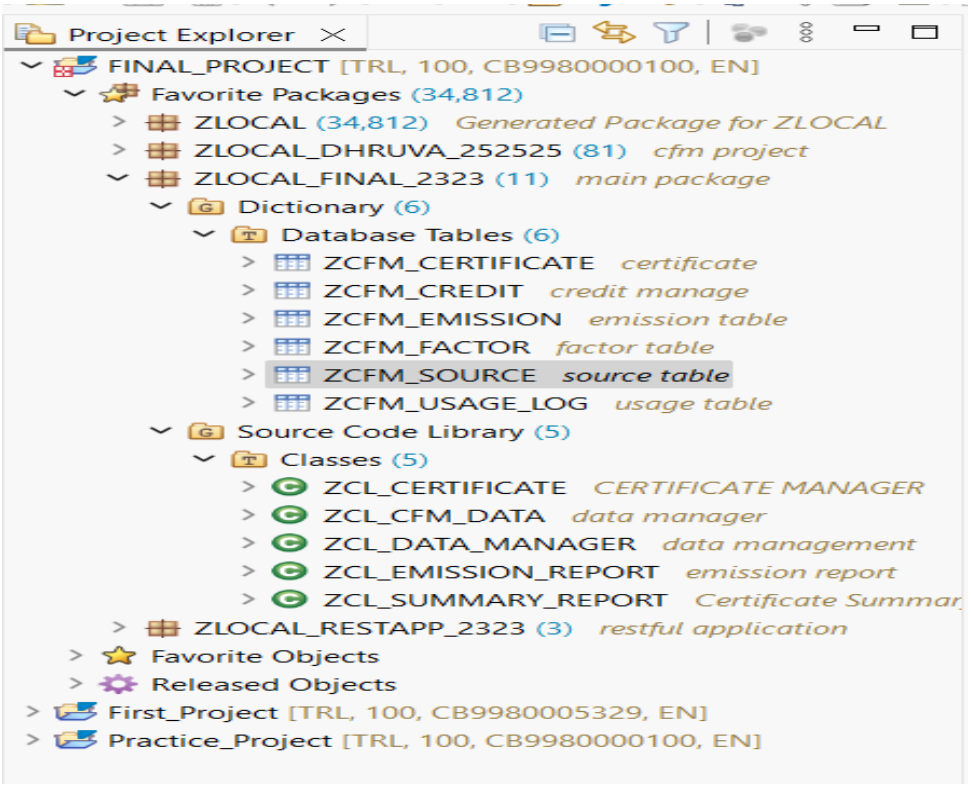
- **Source & Factor Management**: Add, update, and validate emission sources and their factors.
- **Usage Logging**: Store validated energy usage logs with user attribution.
- **Emission Calculation**: Auto-compute CO_2 emissions using factor-based formulas.
- **Credit Assignment**: Classify emissions and assign credits dynamically.
- **Certificate Issuance**: Create carbon certificates with expiry, duplicate detection, and status control.
- **Reporting**: Generate emission summaries and certificate audit logs.

Benefits

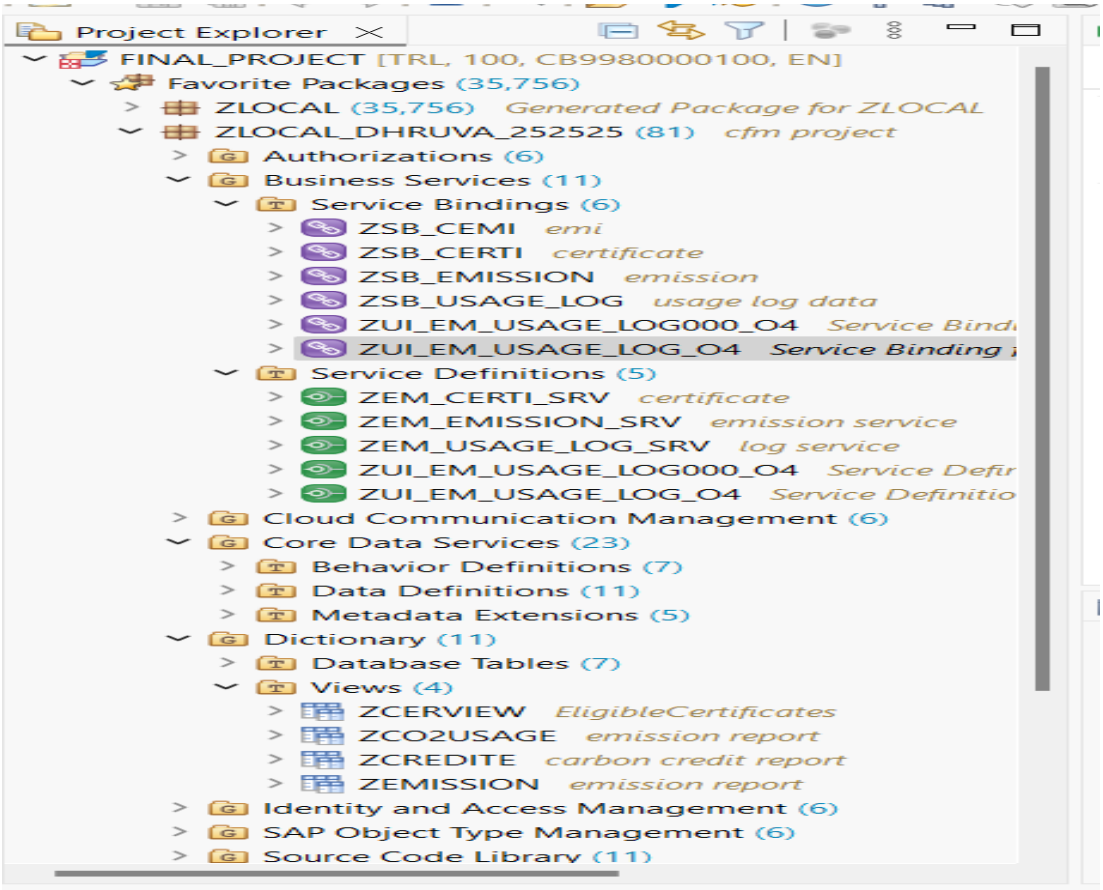
- **Regulatory Compliance**: Helps meet carbon emission reporting standards.
- **Sustainability Focus**: Encourages use of clean energy sources.
- **Data-Driven Decisions**: Provides actionable reports for managers.
- **Integrity & Control**: Validations and structured logging ensure reliable data.

- **SAP Integration**: Built fully in ABAP, easily extendable to Fiori/UI5 or RAP.

Project Architecture :-



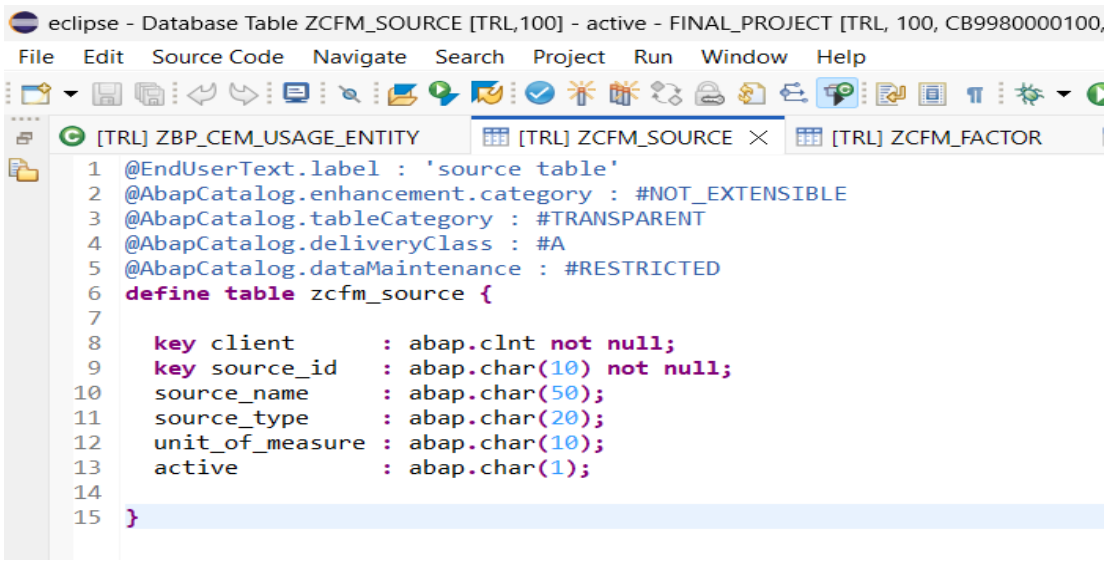
RAP :-

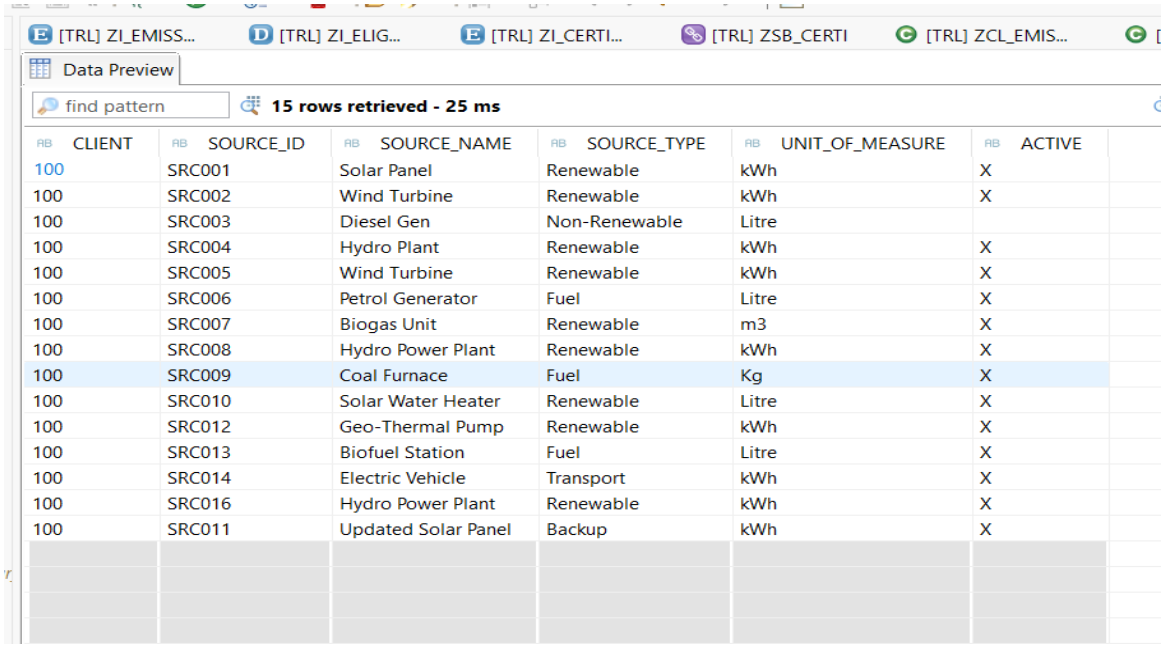


2. Database Schema

2.1 Table: ZCFM_SOURCE – Source Table

Field Name	Data Type	Description
client	abap.clnt	Client Key
source_id	abap.char(10)	Unique Source ID
source_name	abap.char(50)	Source Name
source_type	abap.char(20)	Source Type
unit_of_measure	abap.char(10)	Unit of Measure
active	abap.char(1)	Active Flag (Y/N)

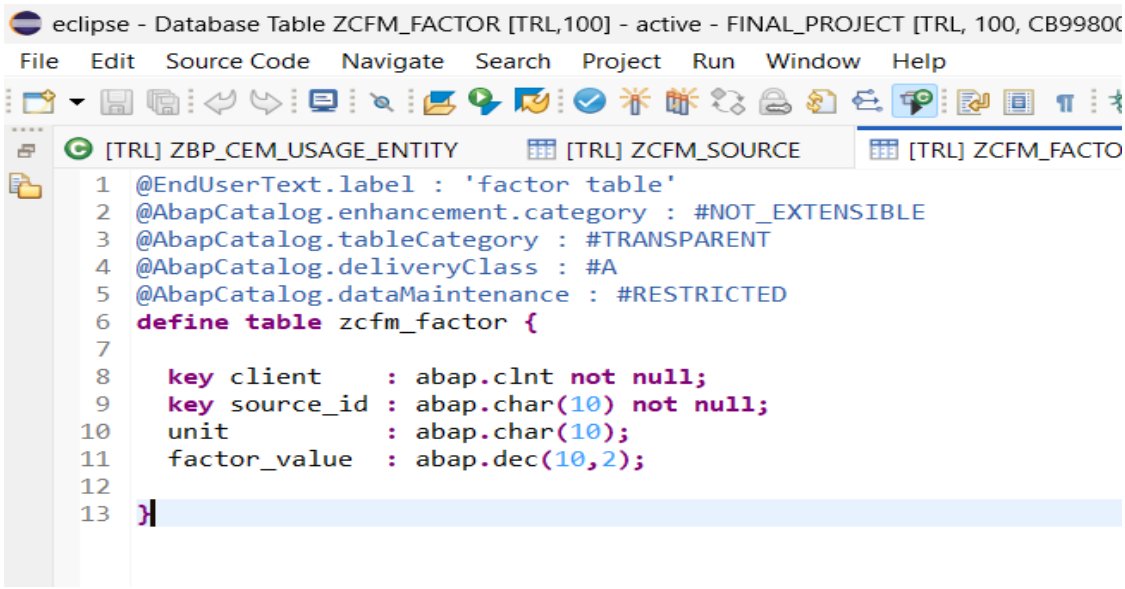


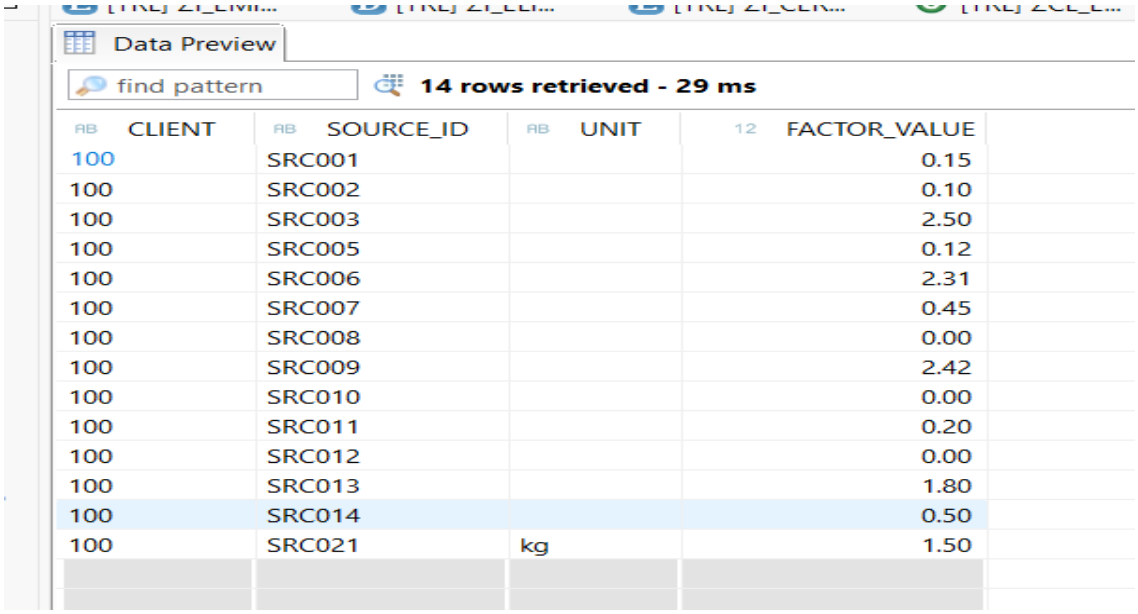


CLIENT	SOURCE_ID	SOURCE_NAME	SOURCE_TYPE	UNIT_OF_MEASURE	ACTIVE
100	SRC001	Solar Panel	Renewable	kWh	X
100	SRC002	Wind Turbine	Renewable	kWh	X
100	SRC003	Diesel Gen	Non-Renewable	Litre	
100	SRC004	Hydro Plant	Renewable	kWh	X
100	SRC005	Wind Turbine	Renewable	kWh	X
100	SRC006	Petrol Generator	Fuel	Litre	X
100	SRC007	Biogas Unit	Renewable	m3	X
100	SRC008	Hydro Power Plant	Renewable	kWh	X
100	SRC009	Coal Furnace	Fuel	Kg	X
100	SRC010	Solar Water Heater	Renewable	Litre	X
100	SRC012	Geo-Thermal Pump	Renewable	kWh	X
100	SRC013	Biofuel Station	Fuel	Litre	X
100	SRC014	Electric Vehicle	Transport	kWh	X
100	SRC016	Hydro Power Plant	Renewable	kWh	X
100	SRC011	Updated Solar Panel	Backup	kWh	X

2.2 Table: ZCFM_FACTOR – Factor Table

Field Name	Data Type	Description
client	abap.clnt	Client Key
source_id	abap.char(10)	Related Source ID
unit	abap.char(10)	Unit of Measurement
factor_value	abap.dec(10,2)	Conversion Factor Value

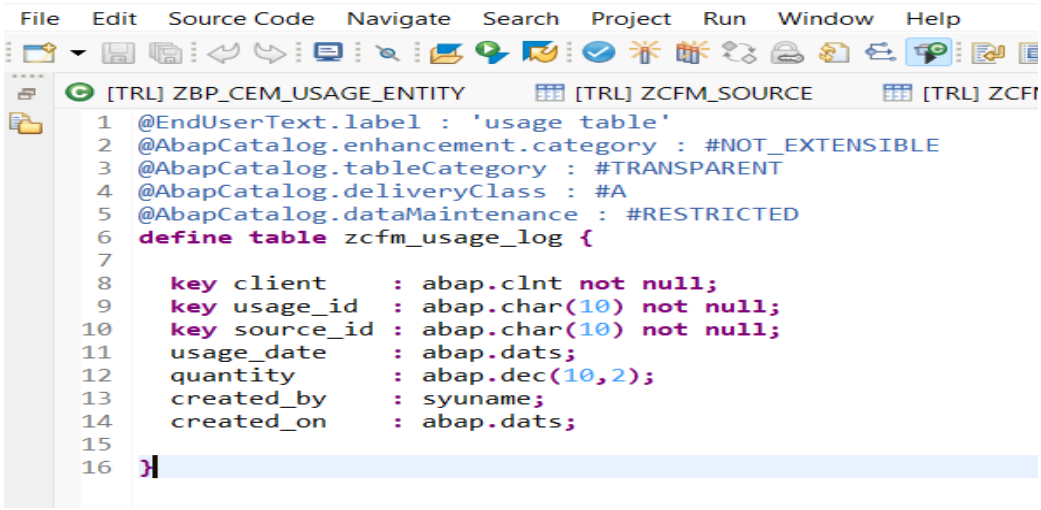




CLIENT	SOURCE_ID	UNIT	FACTOR_VALUE
100	SRC001		0.15
100	SRC002		0.10
100	SRC003		2.50
100	SRC005		0.12
100	SRC006		2.31
100	SRC007		0.45
100	SRC008		0.00
100	SRC009		2.42
100	SRC010		0.00
100	SRC011		0.20
100	SRC012		0.00
100	SRC013		1.80
100	SRC014		0.50
100	SRC021	kg	1.50

2.3 Table: ZCFM_USAGE_LOG – Usage Log Table

Field Name	Data Type	Description
client	abap.clnt	Client Key
usage_id	abap.char(10)	Unique Usage Log ID
source_id	abap.char(10)	Source ID
usage_date	abap.dats	Date of Usage
quantity	abap.dec(10,2)	Quantity Used
created_by	syuname	Created By (User Name)
created_on	abap.dats	Created On (Date)



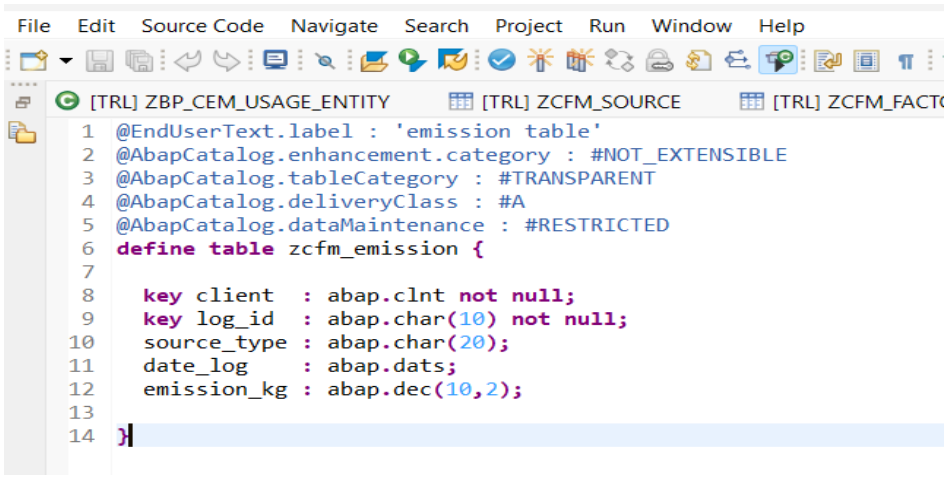
Data Preview

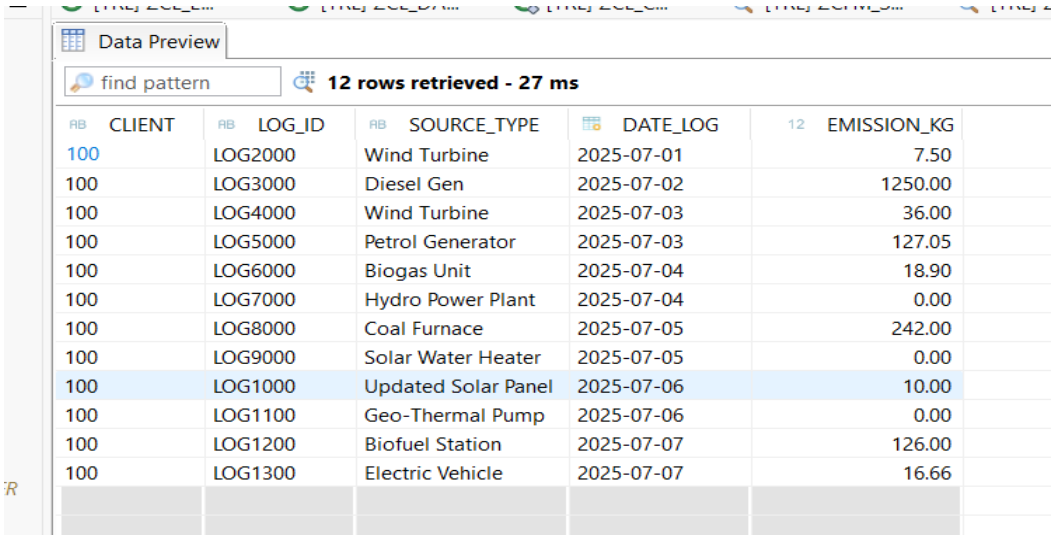
find pattern — 15 rows retrieved - 29 ms

CLIENT	USAGE_ID	SOURCE_ID	USAGE_DATE	QUANTITY	CREATED_BY
100	USG001	SRC001	2025-07-01	120.50	ADMIN
100	USG002	SRC002	2025-07-01	75.00	ADMIN
100	USG003	SRC003	2025-07-02	500.00	DHRUVA
100	USG004	SRC004	2025-07-02	80.00	DHRUVA
100	USG005	SRC005	2025-07-03	300.00	ADMIN
100	USG006	SRC006	2025-07-03	55.00	ADMIN
100	USG007	SRC007	2025-07-04	42.00	DHRUVA
100	USG008	SRC008	2025-07-04	150.00	DHRUVA
100	USG009	SRC009	2025-07-05	100.00	ADMIN
100	USG010	SRC010	2025-07-05	90.00	ADMIN
100	USG011	SRC011	2025-07-06	50.00	ADMIN
100	USG012	SRC012	2025-07-06	60.00	DHRUVA
100	USG013	SRC013	2025-07-07	70.00	ADMIN
100	USG014	SRC014	2025-07-07	33.33	ADMIN
100	USG021	SRC021	2025-07-31	120.50	CB9980000100

2.4 Table: ZCFM_EMISSION – Emission Table

Field Name	Data Type	Description
client	abap.clnt	Client Key
log_id	abap.char(10)	Unique Emission Log ID
source_type	abap.char(20)	Type of Source
date_log	abap.dats	Log Date
emission_kg	abap.dec(10,2)	Emission Quantity (in kg)

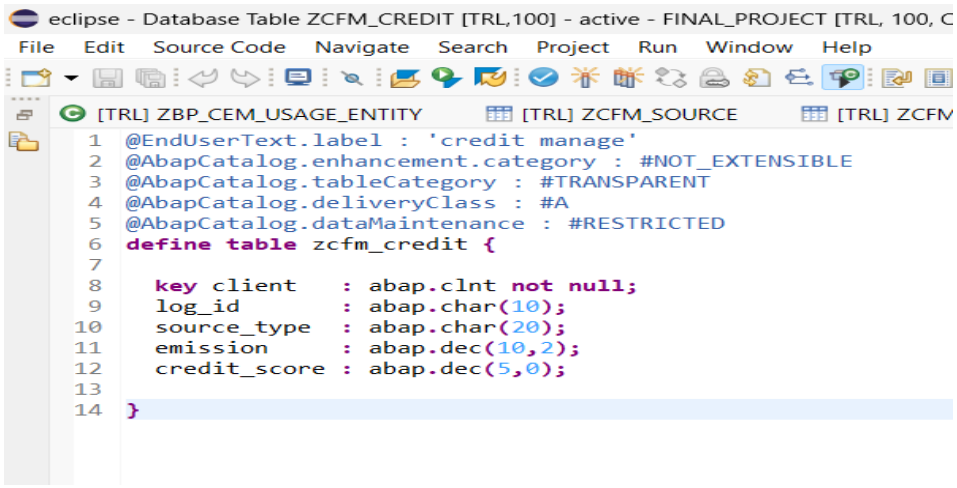




CLIENT	LOG_ID	SOURCE_TYPE	DATE_LOG	EMISSION_KG
100	LOG2000	Wind Turbine	2025-07-01	7.50
100	LOG3000	Diesel Gen	2025-07-02	1250.00
100	LOG4000	Wind Turbine	2025-07-03	36.00
100	LOG5000	Petrol Generator	2025-07-03	127.05
100	LOG6000	Biogas Unit	2025-07-04	18.90
100	LOG7000	Hydro Power Plant	2025-07-04	0.00
100	LOG8000	Coal Furnace	2025-07-05	242.00
100	LOG9000	Solar Water Heater	2025-07-05	0.00
100	LOG1000	Updated Solar Panel	2025-07-06	10.00
100	LOG1100	Geo-Thermal Pump	2025-07-06	0.00
100	LOG1200	Biofuel Station	2025-07-07	126.00
100	LOG1300	Electric Vehicle	2025-07-07	16.66

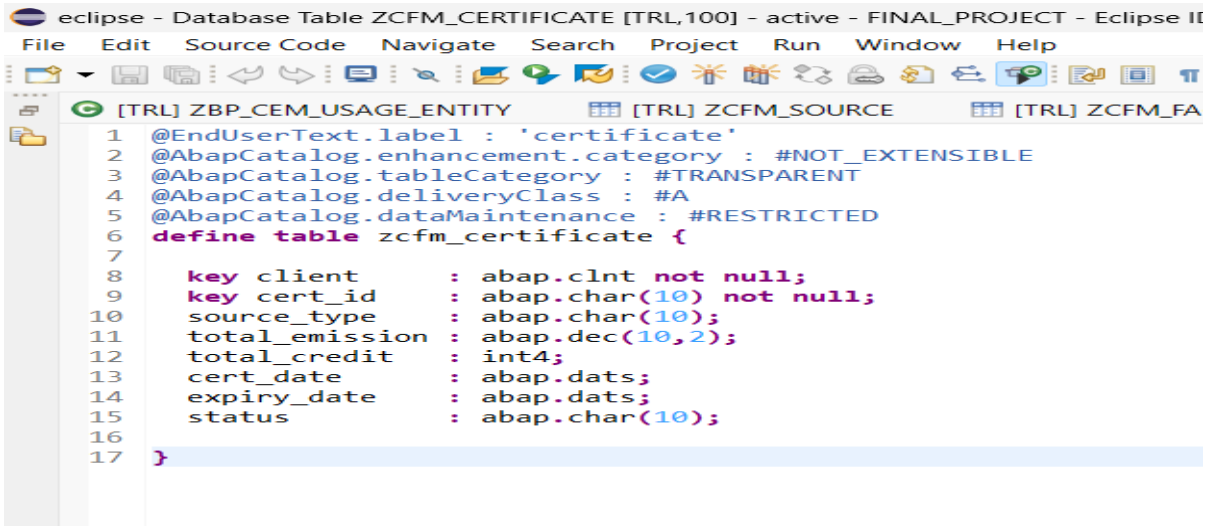
2.5 Table: ZCFM_CREDIT – Credit Score Table

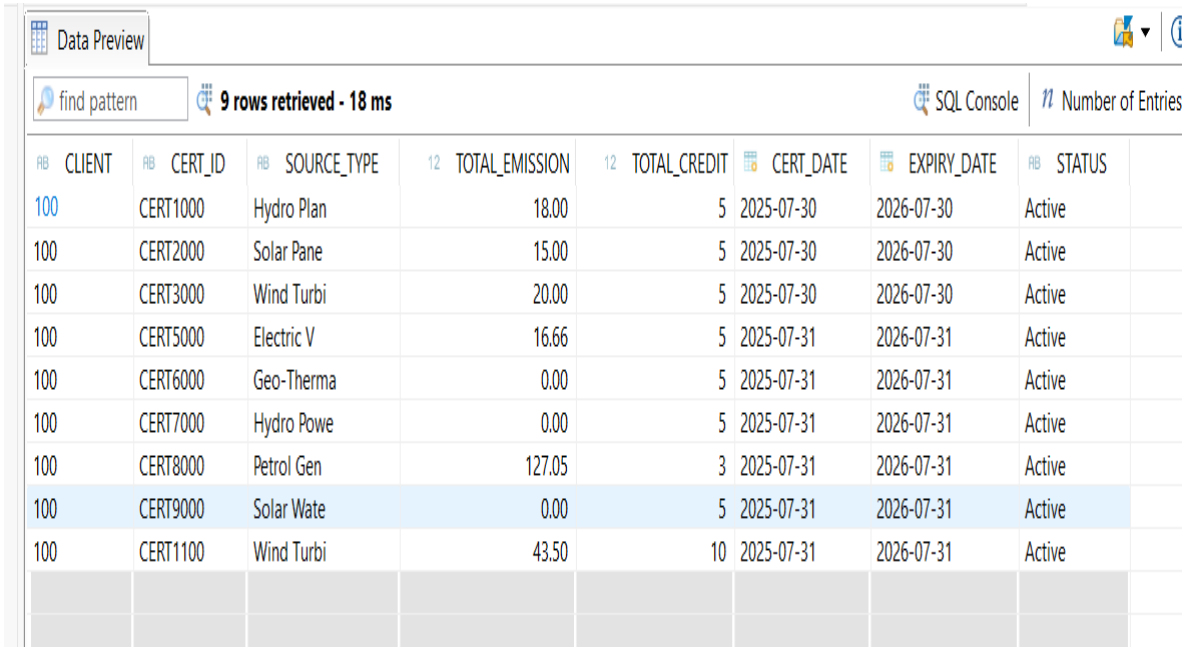
Field Name	Data Type	Description
client	abap.clnt	Client Key
log_id	abap.char(10)	Emission Log Reference ID
source_type	abap.char(20)	Type of Source
emission	abap.dec(10,2)	Total Emission (kg)
credit_score	abap.dec(5,0)	Assigned Credit Score



2.6 Table: ZCFM_CERTIFICATE – Certificate Issue Table

Field Name	Data Type	Description
client	abap.clnt	Client Key
cert_id	abap.char(10)	Certificate ID (Primary Key)
source_type	abap.char(10)	Type of Source
total_emission	abap.dec(10,2)	Total Emission (kg)
total_credit	int4	Total Assigned Credit Score
cert_date	abap.dats	Certificate Issue Date
expiry_date	abap.dats	Certificate Expiry Date
status	abap.char(10)	Certificate Status





CLIENT	CERT_ID	SOURCE_TYPE	TOTAL_EMISSION	TOTAL_CREDIT	CERT_DATE	EXPIRY_DATE	STATUS
100	CERT1000	Hydro Plan	18.00	5	2025-07-30	2026-07-30	Active
100	CERT2000	Solar Pane	15.00	5	2025-07-30	2026-07-30	Active
100	CERT3000	Wind Turbi	20.00	5	2025-07-30	2026-07-30	Active
100	CERT5000	Electric V	16.66	5	2025-07-31	2026-07-31	Active
100	CERT6000	Geo-Therma	0.00	5	2025-07-31	2026-07-31	Active
100	CERT7000	Hydro Powe	0.00	5	2025-07-31	2026-07-31	Active
100	CERT8000	Petrol Gen	127.05	3	2025-07-31	2026-07-31	Active
100	CERT9000	Solar Wate	0.00	5	2025-07-31	2026-07-31	Active
100	CERT1100	Wind Turbi	43.50	10	2025-07-31	2026-07-31	Active

2. Core Functionality (ABAP Classes)

1. CLASS :- zcl_cfm_data
- Handles insertion of core data including sources, emission factors, and usage logs with validation logic.
- Used to initialize or update master and transactional data.

2. CLASS :- zcl_data_manager
- Provides CRUD (Create, Read, Update, Delete) operations specifically for managing source records.
- Supports efficient source maintenance with ABAP methods.

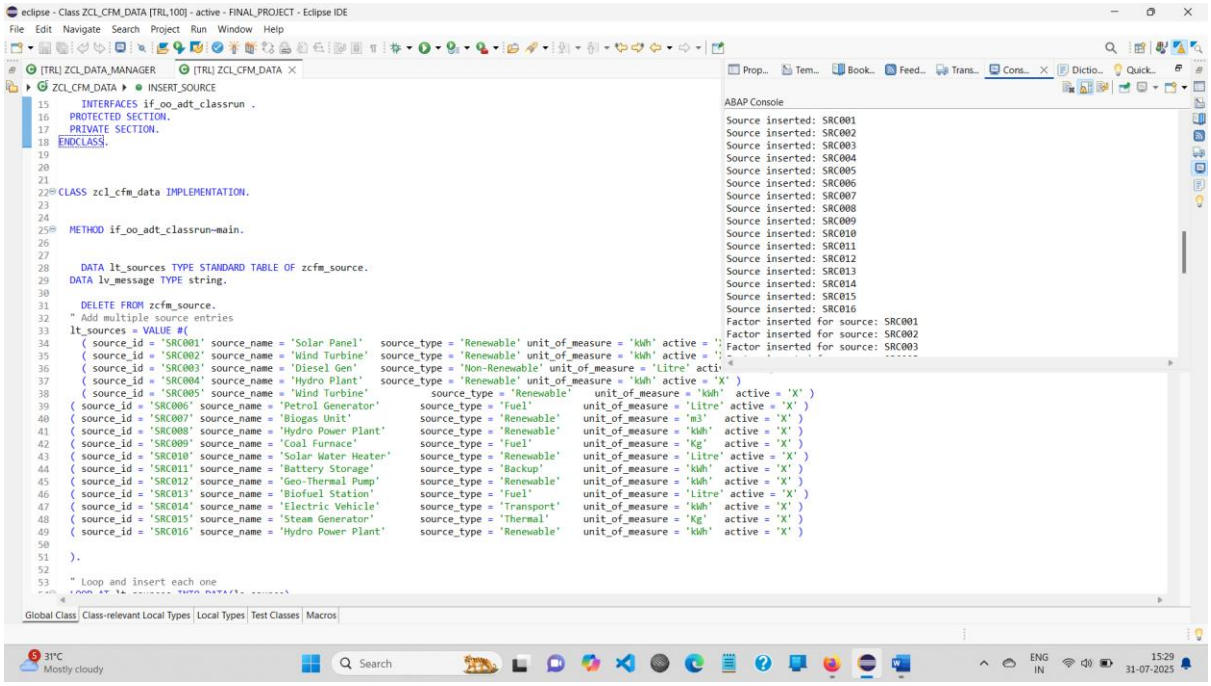
3. CLASS :- zcl_emission_report
- Calculates carbon emissions by combining usage logs and emission factors.
- Stores emission results in a dedicated emission log table (zcfm_emission).

4. CLASS :- zcl_certificate
- Analyzes emissions and assigns carbon credits based on thresholds.
- Generates carbon certificates per source type with expiry control and duplicate checks.
- Logs failed certificate attempts for audit and traceability.

5. CLASS :- zcl_summary_report
- Produces a summarized report of all carbon certificates.
- Shows total issued, expired, and failed certificates in a readable console format.

Class : ZCL_CFM_DATA - Data Manager for Source, Factor, and Usage Logs

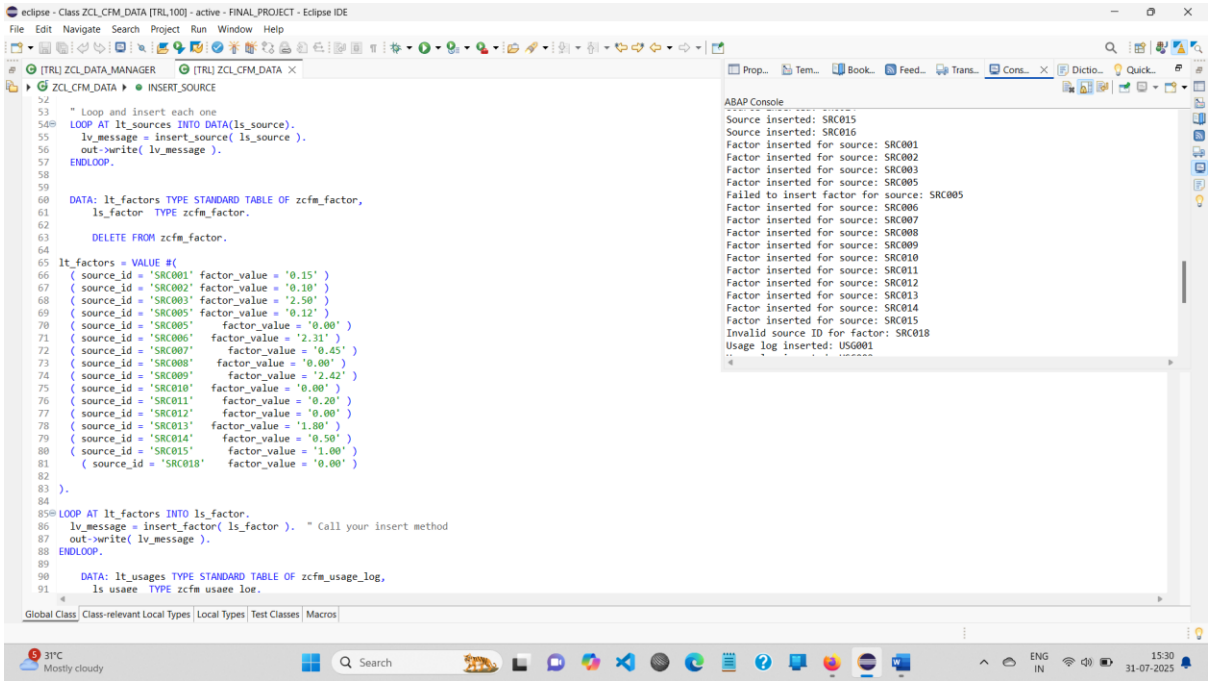
Method : insert_source - Insert new energy source details into ZCFM_SOURCE



```
    INTERFACES if_oo_adt_classrun .
  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.


CLASS zcl_cfm_data IMPLEMENTATION.


  METHOD if_oo_adt_classrun~main.


    DATA lt_sources TYPE STANDARD TABLE OF zcfm_source.
  DATA lv_message TYPE string.

    DELETE FROM zcfm_source.
  " Add multiple source entries
  lt_sources = VALUE #(
    ( source_id = 'SRC001' source_name = 'Solar Panel'   source_type = 'Renewable' unit_of_measure = 'kWh' active = '
    ( source_id = 'SRC002' source_name = 'Wind Turbine'  source_type = 'Renewable' unit_of_measure = 'kWh' active = '
    ( source_id = 'SRC003' source_name = 'Diesel Gen'    source_type = 'Non-Renewable' unit_of_measure = 'Litre' acti
    ( source_id = 'SRC004' source_name = 'Hydro Plant'   source_type = 'Renewable' unit_of_measure = 'kWh' active = 'X' )
    ( source_id = 'SRC005' source_name = 'Wind Turbine'          source_type = 'Renewable'    unit_of_measure = 'kWh'  active = 'X' )
  ( source_id = 'SRC006' source_name = 'Petrol Generator'      source_type = 'Fuel'         unit_of_measure = 'Litre' active = 'X' )
  ( source_id = 'SRC007' source_name = 'Biogas Unit'           source_type = 'Renewable'    unit_of_measure = 'm3'   active = 'X' )
  ( source_id = 'SRC008' source_name = 'Hydro Power Plant'     source_type = 'Renewable'    unit_of_measure = 'kWh'  active = 'X' )
  ( source_id = 'SRC009' source_name = 'Coal Furnace'          source_type = 'Fuel'         unit_of_measure = 'Kg'   active = 'X' )
  ( source_id = 'SRC010' source_name = 'Solar Water Heater'    source_type = 'Renewable'    unit_of_measure = 'Litre' active = 'X' )
  ( source_id = 'SRC011' source_name = 'Battery Storage'       source_type = 'Backup'       unit_of_measure = 'kWh'  active = 'X' )
  ( source_id = 'SRC012' source_name = 'Geo-Thermal Pump'      source_type = 'Renewable'    unit_of_measure = 'kWh'  active = 'X' )
  ( source_id = 'SRC013' source_name = 'Biofuel Station'       source_type = 'Fuel'         unit_of_measure = 'Litre' active = 'X' )
  ( source_id = 'SRC014' source_name = 'Electric Vehicle'      source_type = 'Transport'    unit_of_measure = 'kWh'  active = 'X' )
  ( source_id = 'SRC015' source_name = 'Steam Generator'       source_type = 'Thermal'      unit_of_measure = 'Kg'   active = 'X' )
  ( source_id = 'SRC016' source_name = 'Hydro Power Plant'     source_type = 'Renewable'    unit_of_measure = 'kWh'  active = 'X' )

  ).

  " Loop and insert each one
```

```
ABAP Console
Source inserted: SRC001
Source inserted: SRC002
Source inserted: SRC003
Source inserted: SRC004
Source inserted: SRC005
Source inserted: SRC006
Source inserted: SRC007
Source inserted: SRC008
Source inserted: SRC009
Source inserted: SRC010
Source inserted: SRC011
Source inserted: SRC012
Source inserted: SRC013
Source inserted: SRC014
Source inserted: SRC015
Source inserted: SRC016
Factor inserted for source: SRC001
Factor inserted for source: SRC002
Factor inserted for source: SRC003
```

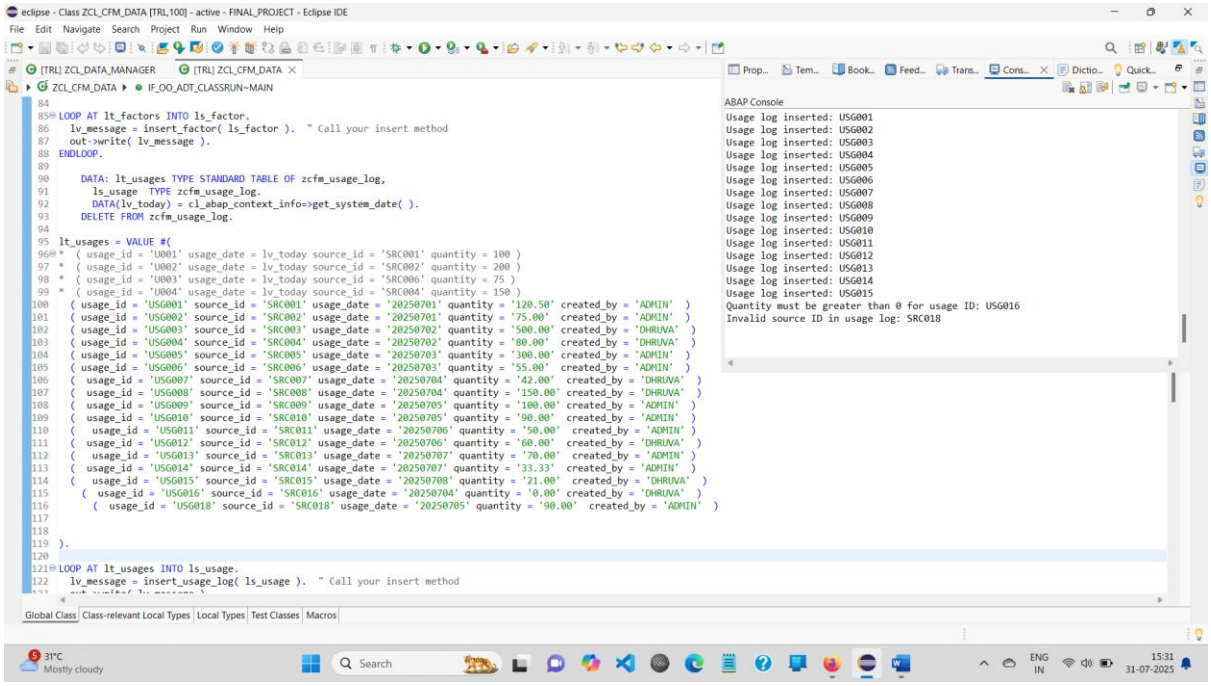
Method : insert_factor - Insert emission factor value linked to source



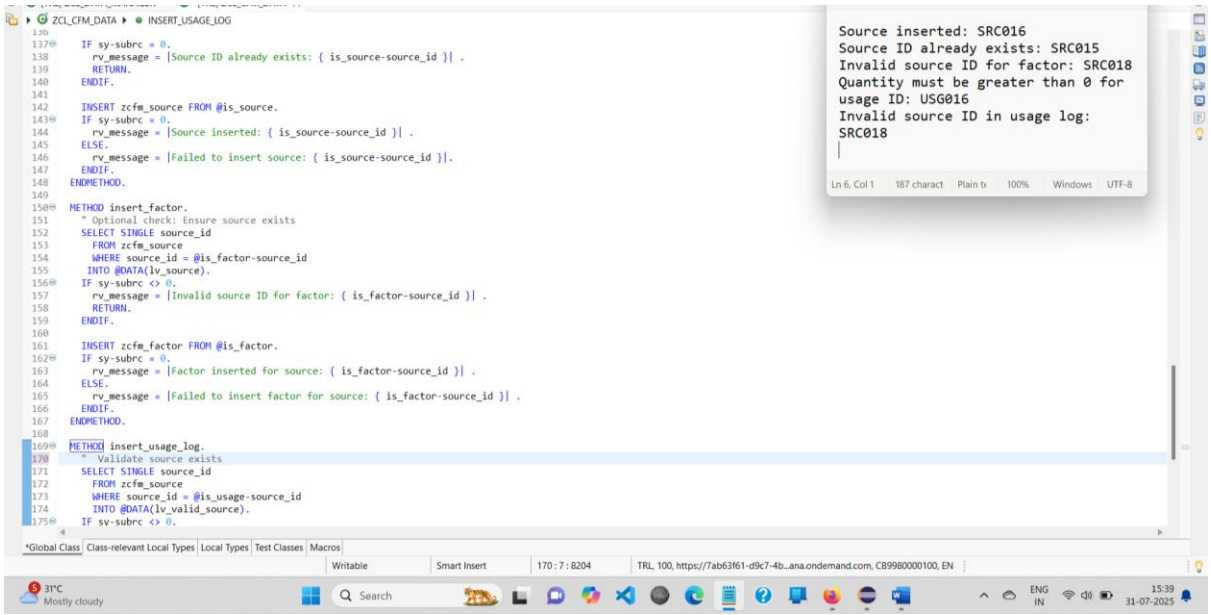
```
  " Loop and insert each one
  LOOP AT lt_sources INTO DATA(ls_source).
    lv_message = insert_source( ls_source ).
    out->write( lv_message ).
  ENDLOOP.


  DATA: lt_factors TYPE STANDARD TABLE OF zcfm_factor,
      ls_factor  TYPE zcfm_factor.

      DELETE FROM zcfm_factor.

lt_factors = VALUE #(
  ( source_id = 'SRC001' factor_value = '0.15' )
  ( source_id = 'SRC002' factor_value = '0.10' )
  ( source_id = 'SRC003' factor_value = '2.50' )
  ( source_id = 'SRC005' factor_value = '0.12' )
  ( source_id = 'SRC005'     factor_value = '0.00' )
  ( source_id = 'SRC006'   factor_value = '2.31' )
  ( source_id = 'SRC007'     factor_value = '0.45' )
  ( source_id = 'SRC008'   factor_value = '0.00' )
  ( source_id = 'SRC009'     factor_value = '2.42' )
  ( source_id = 'SRC010'   factor_value = '0.00' )
  ( source_id = 'SRC011'     factor_value = '0.20' )
  ( source_id = 'SRC012'     factor_value = '0.00' )
  ( source_id = 'SRC013'   factor_value = '1.80' )
  ( source_id = 'SRC014'     factor_value = '0.50' )
  ( source_id = 'SRC015'     factor_value = '1.00' )
  ( source_id = 'SRC018'     factor_value = '0.00' )

).

LOOP AT lt_factors INTO ls_factor.
  lv_message = insert_factor( ls_factor ).  " Call your insert method
  out->write( lv_message ).
ENDLOOP.

    DATA: lt_usages TYPE STANDARD TABLE OF zcfm_usage_log,
      ls_usage  TYPE zcfm_usage_log.
```

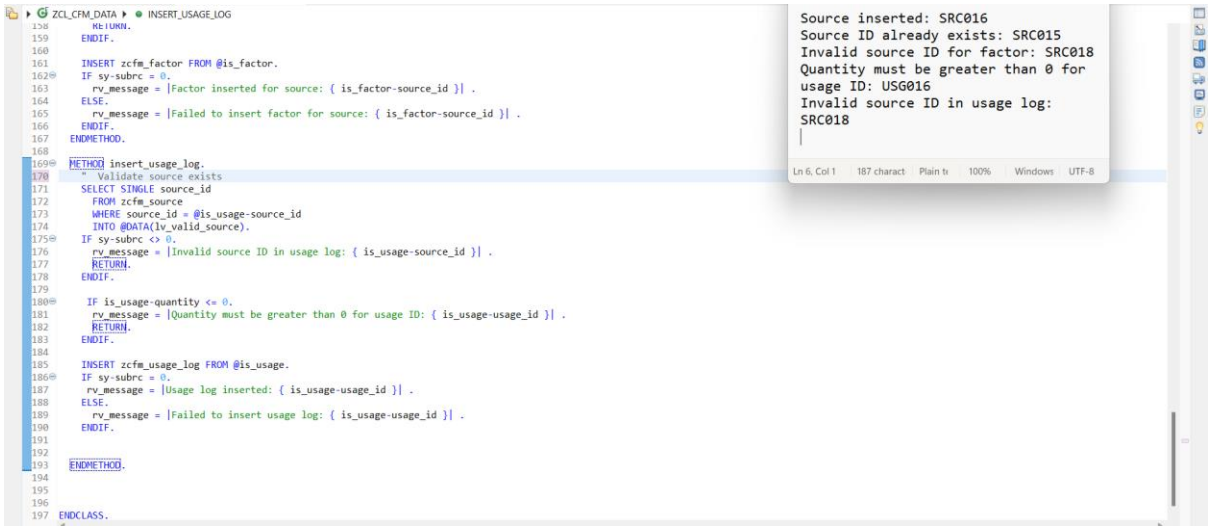
```
ABAP Console
Source inserted: SRC015
Source inserted: SRC016
Factor inserted for source: SRC001
Factor inserted for source: SRC002
Factor inserted for source: SRC003
Factor inserted for source: SRC005
Failed to insert factor for source: SRC005
Factor inserted for source: SRC006
Factor inserted for source: SRC007
Factor inserted for source: SRC008
Factor inserted for source: SRC009
Factor inserted for source: SRC010
Factor inserted for source: SRC011
Factor inserted for source: SRC012
Factor inserted for source: SRC013
Factor inserted for source: SRC014
Factor inserted for source: SRC015
Invalid source ID for factor: SRC018
Usage log inserted: USG001
```

Method : insert_usage_log - Insert usage entry with quantity for energy source



Validation for Energy Source, Factor, and Usage Logs in ABAP



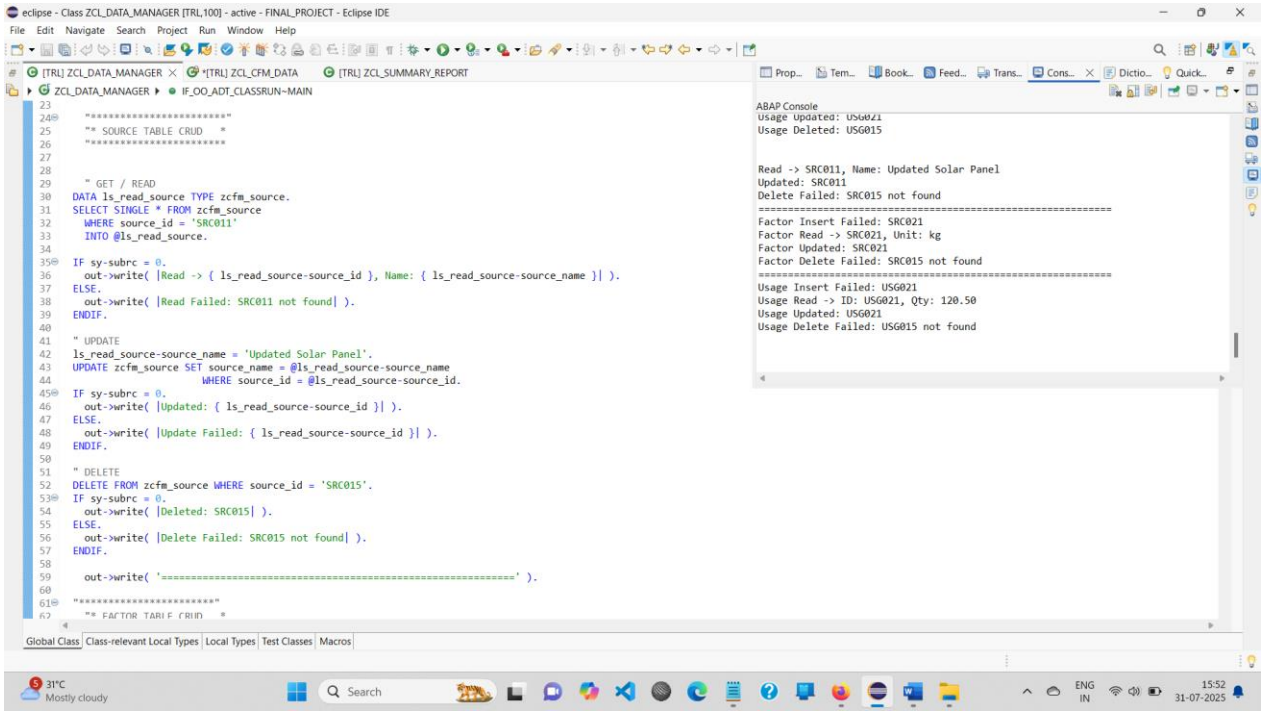


```
    RETURN.
  ENDIF.

  INSERT zcfm_factor FROM @is_factor.
  IF sy-subrc = 0.
    rv_message = |Factor inserted for source: { is_factor-source_id }| .
  ELSE.
    rv_message = |Failed to insert factor for source: { is_factor-source_id }| .
  ENDIF.
ENDMETHOD.

METHOD insert_usage_log.
  " Validate source exists
  SELECT SINGLE source_id
    FROM zcfm_source
    WHERE source_id = @is_usage-source_id
    INTO @DATA(lv_valid_source).
  IF sy-subrc <> 0.
    rv_message = |Invalid source ID in usage log: { is_usage-source_id }| .
    RETURN.
  ENDIF.

   IF is_usage-quantity <= 0.
    rv_message = |Quantity must be greater than 0 for usage ID: { is_usage-usage_id }| .
    RETURN.
  ENDIF.

  INSERT zcfm_usage_log FROM @is_usage.
  IF sy-subrc = 0.
   rv_message = |Usage log inserted: { is_usage-usage_id }| .
  ELSE.
    rv_message = |Failed to insert usage log: { is_usage-usage_id }| .
  ENDIF.


ENDMETHOD.



ENDCLASS.
```

```
Source inserted: SRC016
Source ID already exists: SRC015
Invalid source ID for factor: SRC018
Quantity must be greater than 0 for
usage ID: USG016
Invalid source ID in usage log:
SRC018
```

Class: ZCL_DATA_MANAGER – CRUD Operations for Source, Factor & Usage Logs

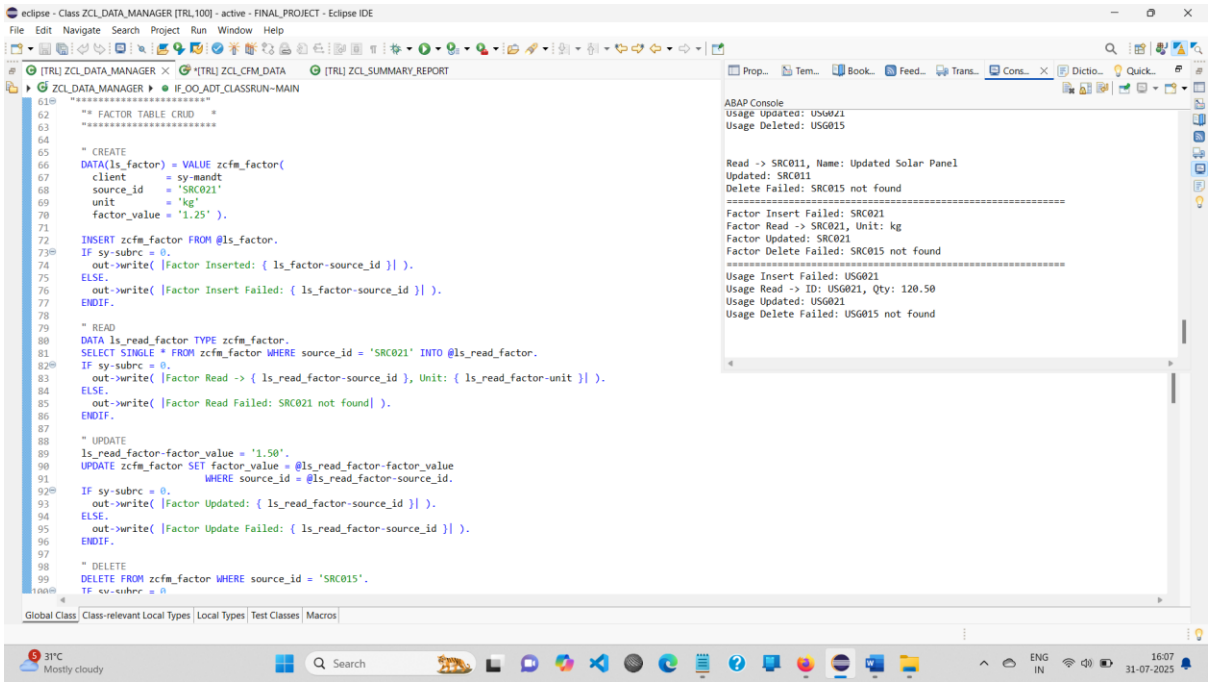
Source Table CRUD Operations :-



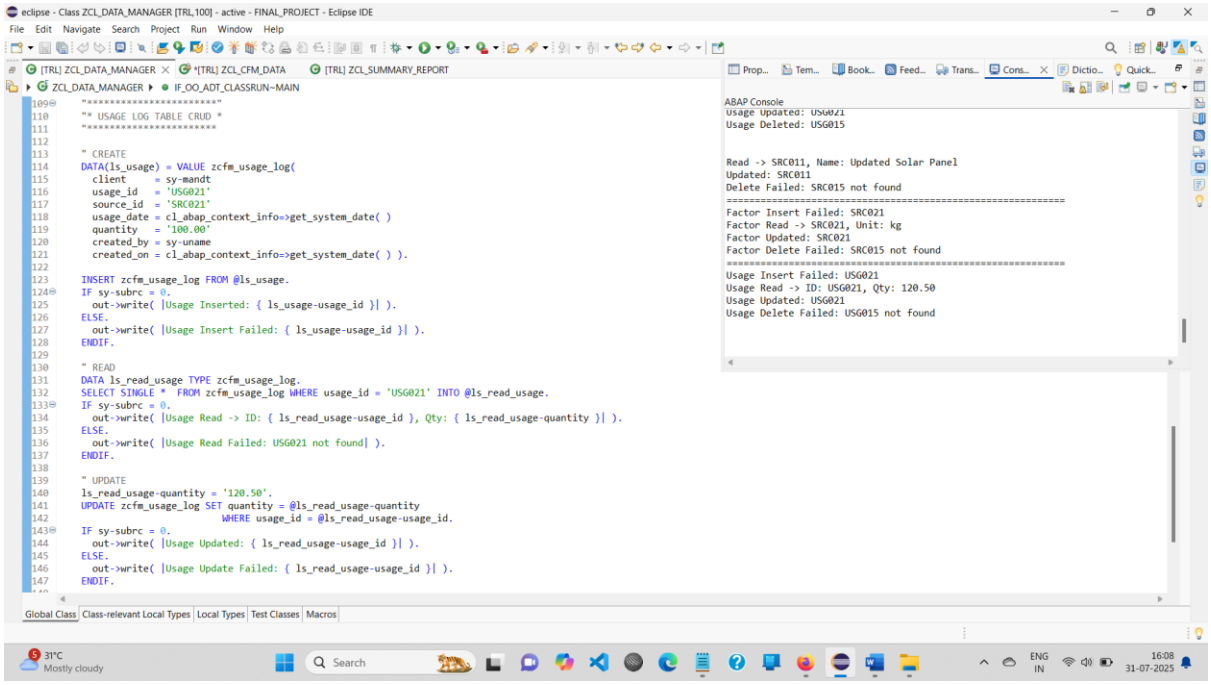
```
  "**********************"
  "* SOURCE TABLE CRUD   *
  "**********************


  " GET / READ
DATA ls_read_source TYPE zcfm_source.
SELECT SINGLE * FROM zcfm_source
  WHERE source_id = 'SRC011'
  INTO @ls_read_source.

IF sy-subrc = 0.
  out->write( |Read -> { ls_read_source-source_id }, Name: { ls_read_source-source_name }| ).
ELSE.
  out->write( |Read Failed: SRC011 not found| ).
ENDIF.

" UPDATE
ls_read_source-source_name = 'Updated Solar Panel'.
UPDATE zcfm_source SET source_name = @ls_read_source-source_name
                   WHERE source_id = @ls_read_source-source_id.
IF sy-subrc = 0.
  out->write( |Updated: { ls_read_source-source_id }| ).
ELSE.
  out->write( |Update Failed: { ls_read_source-source_id }| ).
ENDIF.

" DELETE
DELETE FROM zcfm_source WHERE source_id = 'SRC015'.
IF sy-subrc = 0.
  out->write( |Deleted: SRC015| ).
ELSE.
  out->write( |Delete Failed: SRC015 not found| ).
ENDIF.

  out->write( '============================================================' ).

"**********************
  "* FACTOR TABLE CRUD   *
```

```
ABAP Console
Usage Updated: USG021
Usage Deleted: USG015


Read -> SRC011, Name: Updated Solar Panel
Updated: SRC011
Delete Failed: SRC015 not found
============================================================
Factor Insert Failed: SRC021
Factor Read -> SRC021, Unit: kg
Factor Updated: SRC021
Factor Delete Failed: SRC015 not found
============================================================
Usage Insert Failed: USG021
Usage Read -> ID: USG021, Qty: 120.50
Usage Updated: USG021
Usage Delete Failed: USG015 not found
```

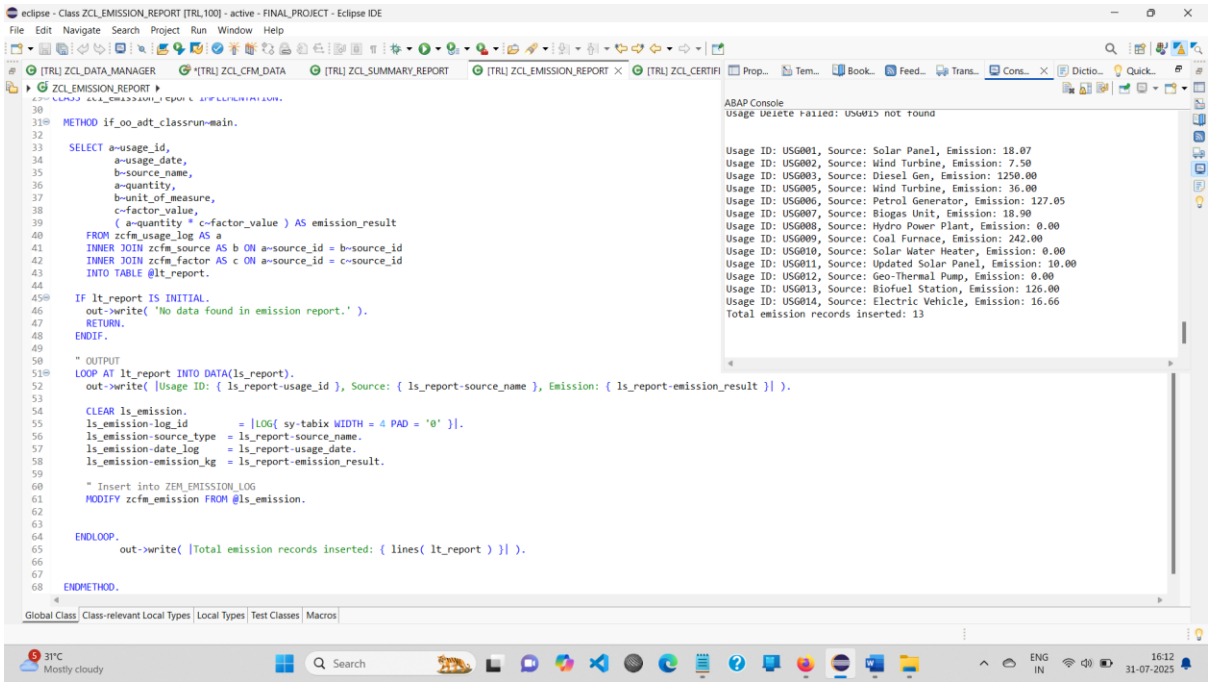
Emission Factor Management :-



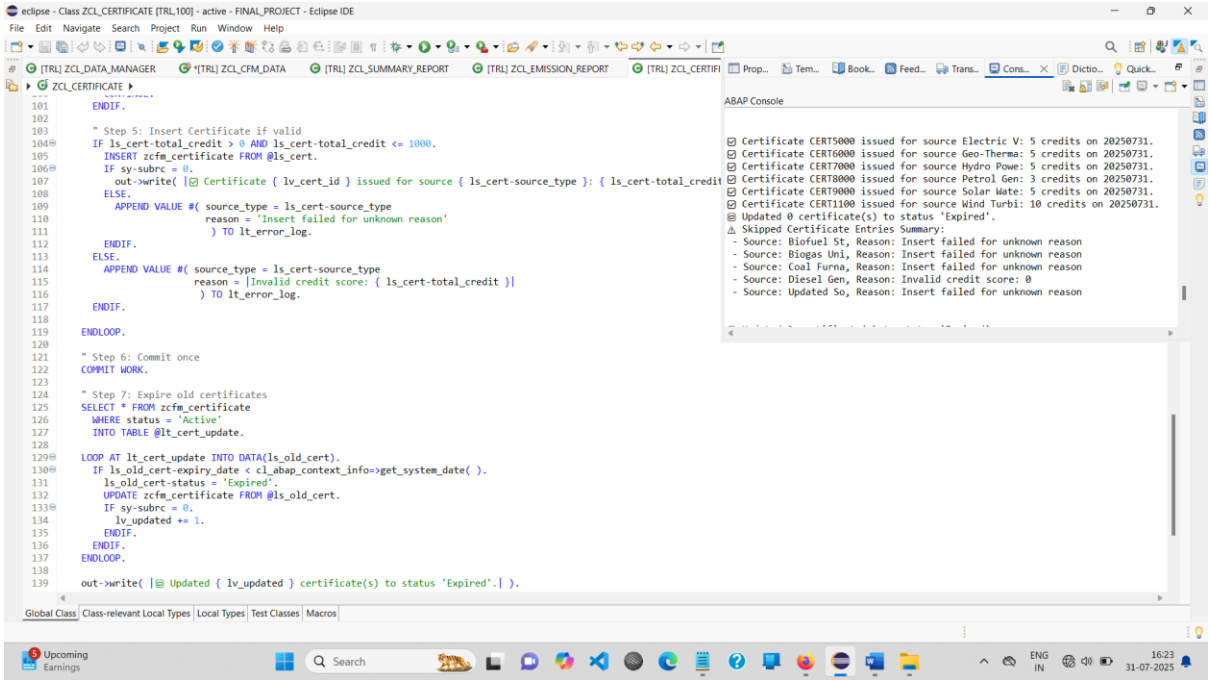
Usage Log Handling :-

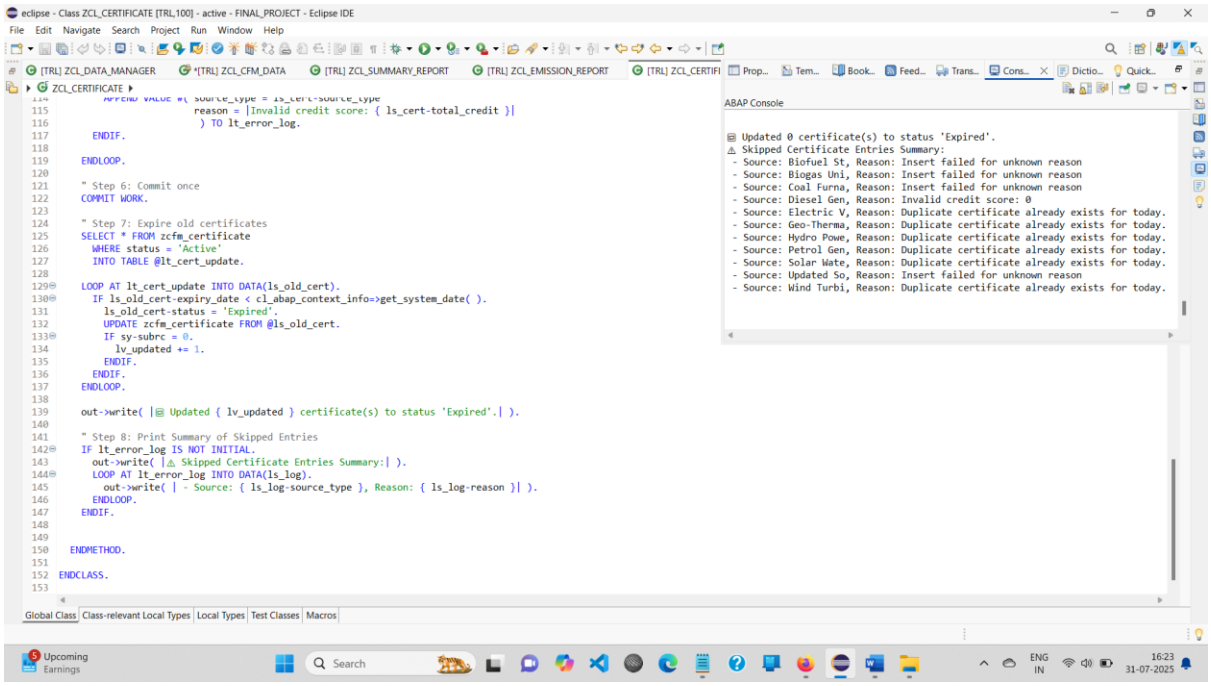


Class : ZCL_EMISSION_REPORT - Emission Report Generator

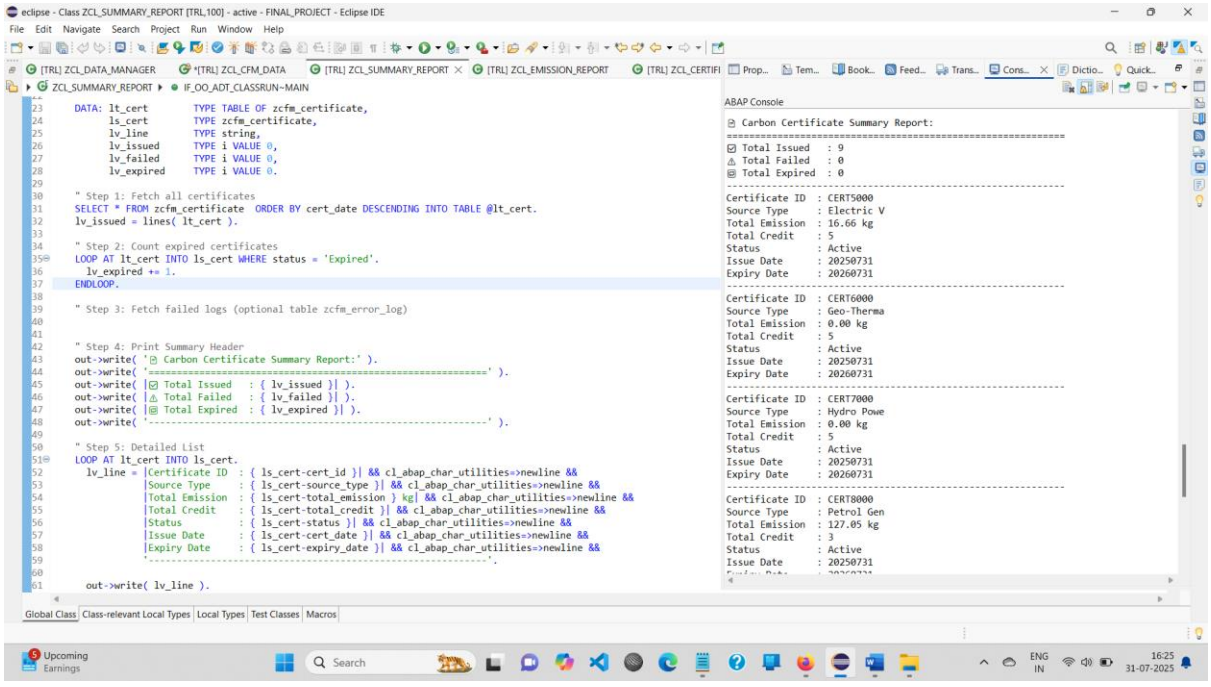


Class : ZCL_CERTIFICATE - Carbon Certificate Issuer





Class : ZCL_SUMMARY_REPORT - Carbon Credit Report Dashboard



CODE :-

CLASS :- ZCL_CFM_DATA

```
CLASS zcl_cfm_data DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC .

  PUBLIC SECTION.
   METHODS:
     insert_source     IMPORTING is_source TYPE zcfm_source
                  RETURNING VALUE(rv_message) TYPE string,
     insert_factor     IMPORTING is_factor TYPE zcfm_factor
                   RETURNING VALUE(rv_message) TYPE string,

     insert_usage_log  IMPORTING is_usage TYPE zcfm_usage_log
                  RETURNING VALUE(rv_message) TYPE string.
    INTERFACES if_oo_adt_classrun .
  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.

CLASS zcl_cfm_data IMPLEMENTATION.

  METHOD if_oo_adt_classrun~main.

    DATA lt_sources TYPE STANDARD TABLE OF zcfm_source.
  DATA lv_message TYPE string.

    DELETE FROM zcfm_source.
  " Add multiple source entries
  lt_sources = VALUE #(
  ( source_id = 'SRC001' source_name = 'Solar Panel'   source_type = 'Renewable'
unit_of_measure = 'kWh' active = 'X' )
  ( source_id = 'SRC002' source_name = 'Wind Turbine'  source_type = 'Renewable'
unit_of_measure = 'kWh' active = 'X' )
  ( source_id = 'SRC003' source_name = 'Diesel Gen'    source_type = 'Non-Renewable'
unit_of_measure = 'Litre' active = ' ' )

  ).

  " Loop and insert each one
  LOOP AT lt_sources INTO DATA(ls_source).
    lv_message = insert_source( ls_source ).
    out->write( lv_message ).
  ENDLOOP.

  DATA: lt_factors TYPE STANDARD TABLE OF zcfm_factor,
     ls_factor  TYPE zcfm_factor.

     DELETE FROM zcfm_factor.

lt_factors = VALUE #(
```

```
  ( source_id = 'SRC001' factor_value = '0.15' )
  ( source_id = 'SRC002' factor_value = '0.10' )
  ( source_id = 'SRC003' factor_value = '2.50' )
).

LOOP AT lt_factors INTO ls_factor.
  lv_message = insert_factor( ls_factor ).  " Call your insert method
  out->write( lv_message ).
ENDLOOP.

    DATA: lt_usages TYPE STANDARD TABLE OF zcfm_usage_log,
      ls_usage  TYPE zcfm_usage_log.
      DATA(lv_today) = cl_abap_context_info=>get_system_date( ).
    DELETE FROM zcfm_usage_log.

lt_usages = VALUE #(

  ( usage_id = 'USG001' source_id = 'SRC001' usage_date = '20250701' quantity = '120.50'
created_by = 'ADMIN'  )
  ( usage_id = 'USG002' source_id = 'SRC002' usage_date = '20250701' quantity = '75.00'
created_by = 'ADMIN'  )
  ( usage_id = 'USG003' source_id = 'SRC003' usage_date = '20250702' quantity = '500.00'
created_by = 'DHRUVA'  )

).

LOOP AT lt_usages INTO ls_usage.
  lv_message = insert_usage_log( ls_usage ).  " Call your insert method
  out->write( lv_message ).
ENDLOOP.

  ENDMETHOD.

  METHOD insert_source.
    " Basic check for existing source
    DATA: lv_exists TYPE zcfm_source-source_id.

    SELECT SINGLE source_id
      FROM zcfm_source
      WHERE source_id = @is_source-source_id
      INTO @lv_exists.

    IF sy-subrc = 0.
      rv_message = |Source ID already exists: { is_source-source_id }| .
      RETURN.
    ENDIF.

    INSERT zcfm_source FROM @is_source.
    IF sy-subrc = 0.
      rv_message = |Source inserted: { is_source-source_id }| .
    ELSE.
      rv_message = |Failed to insert source: { is_source-source_id }|.
    ENDIF.
  ENDMETHOD.
```

```
  METHOD insert_factor.
    " Optional check: Ensure source exists
    SELECT SINGLE source_id
      FROM zcfm_source
      WHERE source_id = @is_factor-source_id
     INTO @DATA(lv_source).
    IF sy-subrc <> 0.
      rv_message = |Invalid source ID for factor: { is_factor-source_id }| .
      RETURN.
    ENDIF.

    INSERT zcfm_factor FROM @is_factor.
    IF sy-subrc = 0.
      rv_message = |Factor inserted for source: { is_factor-source_id }| .
    ELSE.
      rv_message = |Failed to insert factor for source: { is_factor-source_id }| .
    ENDIF.
  ENDMETHOD.

  METHOD insert_usage_log.
    "  Validate source exists
    SELECT SINGLE source_id
      FROM zcfm_source
      WHERE source_id = @is_usage-source_id
      INTO @DATA(lv_valid_source).
    IF sy-subrc <> 0.
      rv_message = |Invalid source ID in usage log: { is_usage-source_id }| .
      RETURN.
    ENDIF.

     IF is_usage-quantity <= 0.
      rv_message = |Quantity must be greater than 0 for usage ID: { is_usage-usage_id }| .
      RETURN.
    ENDIF.

    INSERT zcfm_usage_log FROM @is_usage.
    IF sy-subrc = 0.
     rv_message = |Usage log inserted: { is_usage-usage_id }| .
    ELSE.
      rv_message = |Failed to insert usage log: { is_usage-usage_id }| .
    ENDIF.

  ENDMETHOD.

ENDCLASS.
```

CLASS :- ZCL_DATA_MANAGER

```
CLASS zcl_data_manager DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC .
  PUBLIC SECTION.
    INTERFACES if_oo_adt_classrun .
  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.

CLASS zcl_data_manager IMPLEMENTATION.

  METHOD if_oo_adt_classrun~main.

  DATA lv_message TYPE string.


    "**********************"
    "* SOURCE TABLE CRUD   *
    "**********************
    " GET / READ
  DATA ls_read_source TYPE zcfm_source.
  SELECT SINGLE * FROM zcfm_source
    WHERE source_id = 'SRC011'
    INTO @ls_read_source.

  IF sy-subrc = 0.
    out->write( |Read -> { ls_read_source-source_id }, Name: { ls_read_source-source_name
}| ).
  ELSE.
    out->write( |Read Failed: SRC011 not found| ).
  ENDIF.

  " UPDATE
  ls_read_source-source_name = 'Updated Solar Panel'.
  UPDATE zcfm_source SET source_name = @ls_read_source-source_name
                    WHERE source_id = @ls_read_source-source_id.
  IF sy-subrc = 0.
    out->write( |Updated: { ls_read_source-source_id }| ).
  ELSE.
    out->write( |Update Failed: { ls_read_source-source_id }| ).
  ENDIF.

  " DELETE
  DELETE FROM zcfm_source WHERE source_id = 'SRC015'.
  IF sy-subrc = 0.
    out->write( |Deleted: SRC015| ).
  ELSE.
    out->write( |Delete Failed: SRC015 not found| ).
  ENDIF.
```

```
   out->write(
'================================================================' ).

  "***********************"
   "* FACTOR TABLE CRUD   *
   "***********************

   " CREATE
   DATA(ls_factor) = VALUE zcfm_factor(
    client      = sy-mandt
    source_id   = 'SRC021'
    unit        = 'kg'
    factor_value = '1.25' ).

   INSERT zcfm_factor FROM @ls_factor.
   IF sy-subrc = 0.
    out->write( |Factor Inserted: { ls_factor-source_id }| ).
   ELSE.
    out->write( |Factor Insert Failed: { ls_factor-source_id }| ).
   ENDIF.

   " READ
   DATA ls_read_factor TYPE zcfm_factor.
   SELECT SINGLE * FROM zcfm_factor WHERE source_id = 'SRC021' INTO
@ls_read_factor.
   IF sy-subrc = 0.
    out->write( |Factor Read -> { ls_read_factor-source_id }, Unit: { ls_read_factor-unit }| ).
   ELSE.
    out->write( |Factor Read Failed: SRC021 not found| ).
   ENDIF.

   " UPDATE
   ls_read_factor-factor_value = '1.50'.
   UPDATE zcfm_factor SET factor_value = @ls_read_factor-factor_value
                    WHERE source_id = @ls_read_factor-source_id.
   IF sy-subrc = 0.
    out->write( |Factor Updated: { ls_read_factor-source_id }| ).
   ELSE.
    out->write( |Factor Update Failed: { ls_read_factor-source_id }| ).
   ENDIF.

   " DELETE
   DELETE FROM zcfm_factor WHERE source_id = 'SRC015'.
   IF sy-subrc = 0.
    out->write( |Factor Deleted: SRC015| ).
   ELSE.
    out->write( |Factor Delete Failed: SRC015 not found| ).
   ENDIF.

      out->write(
'================================================================' ).

   "***********************"
   "* USAGE LOG TABLE CRUD *
   "***********************
```

```
    " CREATE
    DATA(ls_usage) = VALUE zcfm_usage_log(
      client     = sy-mandt
      usage_id   = 'USG021'
      source_id  = 'SRC021'
      usage_date = cl_abap_context_info=>get_system_date( )
      quantity   = '100.00'
      created_by = sy-uname
      created_on = cl_abap_context_info=>get_system_date( ) ).

    INSERT zcfm_usage_log FROM @ls_usage.
    IF sy-subrc = 0.
      out->write( |Usage Inserted: { ls_usage-usage_id }| ).
    ELSE.
      out->write( |Usage Insert Failed: { ls_usage-usage_id }| ).
    ENDIF.

    " READ
    DATA ls_read_usage TYPE zcfm_usage_log.
    SELECT SINGLE *  FROM zcfm_usage_log WHERE usage_id = 'USG021' INTO
@ls_read_usage.
    IF sy-subrc = 0.
      out->write( |Usage Read -> ID: { ls_read_usage-usage_id }, Qty: { ls_read_usage-
quantity }| ).
    ELSE.
      out->write( |Usage Read Failed: USG021 not found| ).
    ENDIF.

    " UPDATE
    ls_read_usage-quantity = '120.50'.
    UPDATE zcfm_usage_log SET quantity = @ls_read_usage-quantity
                          WHERE usage_id = @ls_read_usage-usage_id.
    IF sy-subrc = 0.
      out->write( |Usage Updated: { ls_read_usage-usage_id }| ).
    ELSE.
      out->write( |Usage Update Failed: { ls_read_usage-usage_id }| ).
    ENDIF.

    " DELETE
    DELETE FROM zcfm_usage_log WHERE usage_id = 'USG015'.
    IF sy-subrc = 0.
      out->write( |Usage Deleted: USG015| ).
    ELSE.
      out->write( |Usage Delete Failed: USG015 not found| ).
    ENDIF.

  ENDMETHOD.

ENDCLASS.
```

CLASS :- ZCL_EMISSION_REPORT

```
CLASS zcl_emission_report DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC .

  PUBLIC SECTION.

    INTERFACES if_oo_adt_classrun .

  PROTECTED SECTION.
  PRIVATE SECTION.
   TYPES: BEGIN OF ty_emission,
            usage_id       TYPE zcfm_usage_log-usage_id,
            usage_date     TYPE zcfm_usage_log-usage_date,
            source_name    TYPE zcfm_source-source_name,
            quantity       TYPE zcfm_usage_log-quantity,
            unit           TYPE zcfm_source-unit_of_measure,
            factor_value   TYPE zcfm_factor-factor_value,
            emission_result TYPE p LENGTH 10 DECIMALS 2,
          END OF ty_emission.

    DATA: lt_report TYPE TABLE OF ty_emission,
          ls_emission TYPE zcfm_emission.

ENDCLASS.

CLASS zcl_emission_report IMPLEMENTATION.

  METHOD if_oo_adt_classrun~main.

    SELECT a~usage_id,
           a~usage_date,
           b~source_name,
           a~quantity,
           b~unit_of_measure,
           c~factor_value,
           ( a~quantity * c~factor_value ) AS emission_result
      FROM zcfm_usage_log AS a
      INNER JOIN zcfm_source AS b ON a~source_id = b~source_id
      INNER JOIN zcfm_factor AS c ON a~source_id = c~source_id
      INTO TABLE @lt_report.

    IF lt_report IS INITIAL.
      out->write( 'No data found in emission report.' ).
      RETURN.
    ENDIF.

    " OUTPUT
    LOOP AT lt_report INTO DATA(ls_report).
      out->write( |Usage ID: { ls_report-usage_id }, Source: { ls_report-source_name },
Emission: { ls_report-emission_result }| ).

      CLEAR ls_emission.
```

```
      ls_emission-log_id        = |LOG{ sy-tabix WIDTH = 4 PAD = '0' }|.
      ls_emission-source_type  = ls_report-source_name.
      ls_emission-date_log     = ls_report-usage_date.
      ls_emission-emission_kg  = ls_report-emission_result.

      " Insert into ZEM_EMISSION_LOG
      MODIFY zcfm_emission FROM @ls_emission.

    ENDLOOP.
         out->write( |Total emission records inserted: { lines( lt_report ) }| ).

  ENDMETHOD.
ENDCLASS.
```

CLASS :- ZCL_CERTIFICATE

```
CLASS zcl_certificate DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC.

  PUBLIC SECTION.
    INTERFACES if_oo_adt_classrun.
  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.
CLASS zcl_certificate IMPLEMENTATION.

  METHOD if_oo_adt_classrun~main.

    DATA: lt_emission      TYPE TABLE OF zcfm_emission,
          ls_emission      TYPE zcfm_emission,
          ls_credit        TYPE zcfm_credit,
          lv_index         TYPE i VALUE 1,
          lv_cert_id       TYPE c LENGTH 10,
          ls_cert          TYPE zcfm_certificate,
          lt_credit        TYPE TABLE OF zcfm_credit,
          lt_cert_update   TYPE TABLE OF zcfm_certificate,
          lv_updated       TYPE i VALUE 0.

    " For error logging
    TYPES: BEGIN OF ty_error_log,
             source_type TYPE zcfm_certificate-source_type,
             reason      TYPE string,
             timestamp   TYPE string,
           END OF ty_error_log.

    DATA: lt_error_log TYPE TABLE OF ty_error_log.

    " Step 1: Read Emission Log
    SELECT * FROM zcfm_emission INTO TABLE @lt_emission.

    LOOP AT lt_emission INTO ls_emission.
```

```
    CLEAR ls_credit.
    ls_credit-log_id      = ls_emission-log_id.
    ls_credit-source_type = ls_emission-source_type.
    ls_credit-emission    = ls_emission-emission_kg.

    " Step 2: Assign carbon credit
    IF ls_emission-emission_kg <= 100.
      ls_credit-credit_score = 5.
    ELSEIF ls_emission-emission_kg <= 500.
      ls_credit-credit_score = 3.
    ELSEIF ls_emission-emission_kg <= 1000.
      ls_credit-credit_score = 1.
    ELSE.
      ls_credit-credit_score = 0.
    ENDIF.

    " Step 3: Insert credit entry
    APPEND ls_credit TO lt_credit.
    INSERT zcfm_credit FROM @ls_credit.
  ENDLOOP.

  SORT lt_credit BY source_type.

  " Step 4: Group by source_type to generate certificates
  LOOP AT lt_credit INTO DATA(ls_grouped) GROUP BY ( source_type = ls_grouped-
source_type ) INTO DATA(group).

    CLEAR ls_cert.
    lv_cert_id = |CERT{ lv_index WIDTH = 4 PAD = '0' }|.
    lv_index += 1.

    ls_cert-cert_id     = lv_cert_id.
    ls_cert-source_type = group-source_type.
    ls_cert-cert_date   = cl_abap_context_info=>get_system_date( ).
    ls_cert-expiry_date = cl_abap_context_info=>get_system_date( ) + 365.
    IF ls_cert-expiry_date < cl_abap_context_info=>get_system_date( ) .
  ls_cert-status = 'Expired'.
 ELSE.
  ls_cert-status = 'Active'.
 ENDIF.

    LOOP AT GROUP group ASSIGNING FIELD-SYMBOL(<credit_entry>).
      ls_cert-total_emission += <credit_entry>-emission.
      ls_cert-total_credit   += <credit_entry>-credit_score.
    ENDLOOP.

    " Step 4.1: Check if certificate already exists for today
    SELECT SINGLE cert_id FROM zcfm_certificate
      WHERE source_type = @ls_cert-source_type
        AND cert_date = @ls_cert-cert_date
      INTO @DATA(lv_existing_cert_id).

    IF sy-subrc = 0.
      APPEND VALUE #( source_type = ls_cert-source_type
```

```
                reason = 'Duplicate certificate already exists for today.'
                ) TO lt_error_log.
        CONTINUE.
      ENDIF.

      " Step 5: Insert Certificate if valid
      IF ls_cert-total_credit > 0 AND ls_cert-total_credit <= 1000.
        INSERT zcfm_certificate FROM @ls_cert.
        IF sy-subrc = 0.
          out->write( |☑ Certificate { lv_cert_id } issued for source { ls_cert-source_type }: {
ls_cert-total_credit } credits on { ls_cert-cert_date }.| ).
        ELSE.
          APPEND VALUE #( source_type = ls_cert-source_type
                  reason = 'Insert failed for unknown reason'
                  ) TO lt_error_log.
        ENDIF.
      ELSE.
        APPEND VALUE #( source_type = ls_cert-source_type
                reason = |Invalid credit score: { ls_cert-total_credit }|
                ) TO lt_error_log.
      ENDIF.

    ENDLOOP.

    " Step 6: Commit once
    COMMIT WORK.

    " Step 7: Expire old certificates
    SELECT * FROM zcfm_certificate
      WHERE status = 'Active'
      INTO TABLE @lt_cert_update.

    LOOP AT lt_cert_update INTO DATA(ls_old_cert).
      IF ls_old_cert-expiry_date < cl_abap_context_info=>get_system_date( ).
        ls_old_cert-status = 'Expired'.
        UPDATE zcfm_certificate FROM @ls_old_cert.
        IF sy-subrc = 0.
          lv_updated += 1.
        ENDIF.
      ENDIF.
    ENDLOOP.
    out->write( |🔁 Updated { lv_updated } certificate(s) to status 'Expired'.| ).

    " Step 8: Print Summary of Skipped Entries
    IF lt_error_log IS NOT INITIAL.
      out->write( |⚠ Skipped Certificate Entries Summary:| ).
      LOOP AT lt_error_log INTO DATA(ls_log).
        out->write( | - Source: { ls_log-source_type }, Reason: { ls_log-reason }| ).
      ENDLOOP.
    ENDIF.

  ENDMETHOD.

ENDCLASS.
```

CLASS :- ZCL_SUMMARY_REPORT

```
CLASS zcl_summary_report DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC .

  PUBLIC SECTION.


    INTERFACES if_oo_adt_classrun .
  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.

CLASS zcl_summary_report IMPLEMENTATION.

  METHOD if_oo_adt_classrun~main.

    DATA: lt_cert       TYPE TABLE OF zcfm_certificate,
          ls_cert       TYPE zcfm_certificate,
          lv_line       TYPE string,
          lv_issued     TYPE i VALUE 0,
          lv_failed     TYPE i VALUE 0,
          lv_expired    TYPE i VALUE 0.

    " Step 1: Fetch all certificates

    SELECT * FROM zcfm_certificate  ORDER BY cert_date DESCENDING INTO TABLE
@lt_cert.
    lv_issued = lines( lt_cert ).


    " Step 2: Count expired certificates
    LOOP AT lt_cert INTO ls_cert WHERE status = 'Expired'.
      lv_expired += 1.
    ENDLOOP.


    " Step 4: Print Summary Header
    out->write( '📄 Carbon Certificate Summary Report:' ).
    out->write(
'==========================================================' ).
    out->write( |☑ Total Issued   : { lv_issued }| ).
    out->write( |⚠ Total Failed   : { lv_failed }| ).
    out->write( |🔁 Total Expired  : { lv_expired }| ).
    out->write( '----------------------------------------------------' ).
```

```
    " Step 5: Detailed List
    LOOP AT lt_cert INTO ls_cert.
      lv_line = |Certificate ID  : { ls_cert-cert_id }| && cl_abap_char_utilities=>newline &&
                |Source Type     : { ls_cert-source_type }| && cl_abap_char_utilities=>newline &&
                |Total Emission  : { ls_cert-total_emission } kg| &&
cl_abap_char_utilities=>newline &&
                |Total Credit    : { ls_cert-total_credit }| && cl_abap_char_utilities=>newline &&
                |Status          : { ls_cert-status }| && cl_abap_char_utilities=>newline &&
                |Issue Date      : { ls_cert-cert_date }| && cl_abap_char_utilities=>newline &&
                |Expiry Date     : { ls_cert-expiry_date }| && cl_abap_char_utilities=>newline &&
                '------------------------------------------------------------'.

      out->write( lv_line ).
    ENDLOOP.

  ENDMETHOD.
ENDCLASS.
```

RESTFUL APPLICATION (RAP)

5. Project Description

Overview

The **Carbon Emission Management System (CEMS)** is a backend solution developed using the **RESTful ABAP Programming Model (RAP)**. It enables organizations to monitor their energy usage, calculate carbon emissions, and generate carbon credit certificates automatically based on usage data. The system is fully built on CDS views, RAP behavior classes, and ABAP logic to ensure automation, scalability, and data integrity.
It allows users to manage energy sources, define emission factors, log usage, compute emissions, and certify low-carbon activities with credit scores — supporting sustainable operations and environmental compliance.

Key Features

- **Source Management**: Master data of energy sources (like solar, fuel, grid) stored in ZEM_SOURCES
- **Factor Table**: CO_2 emission factors per unit defined in ZEM_FACTOR_TABLE.
- **Usage Logging**: : All source usage data is captured in ZEM_USAGE_LOG.
- **Emission Calculation**: ZEM_EMISSION CDS view calculates total emission as quantity × co2_factor.
- **Credit Calculation**: ZI_CREDIT_REPORT computes credit scores for emissions below 1000.
- **Certificate Generation**: Certificates are stored in ZEM_CERTIFICATE with status (Active/Expired).
- **Eligibility View**: ZI_ELIGCERTIFICATE shows all issued certificates.

Benefits

- **Automated Carbon Tracking**: No manual calculations—everything is computed via RAP and CDS logic
- **Promotes Sustainability**: Rewards lower emissions with higher credit scores.
- **Validated Entries**: Behaviour classes ensure only valid data is inserted.
- **RAP-Based Architecture**: Clean, modern ABAP code with service binding and OData-readiness.

- **Integration Ready**: Can be extended to Fiori apps or third-party dashboards.

7. **CDS Views (Data Model Layer)**
   7.1. Data Definitions
      7.1.1. ***ZEM_USAGE_ENTITY*** *: Root entity view for energy usage logs. Stores usage data with quantity, date, and source reference.*

      7.1.2. ***ZEM_EMISSION*** *: Calculates total carbon emissions by joining usage data with source and emission factor tables. Computes total_emission = quantity × co2_factor.*

      7.1.3. ***ZI_CREDIT_REPORT*** *: Computes carbon credit scores based on emission levels. Credits are given when total_emission ≤ 1000, calculated dynamically.*

      7.1.4. ***ZI_ELIGCERTIFICATE*** *: Lists all eligible carbon credit certificates issued, including emissions, scores, and validity dates.*

   7.2. *Metadata Extensions (UI)*
      7.2.1. ZC_EM_USAGE_LOG000 : UI projection of usage log data with fields like UsageId, SourceId, UsageDate, and Quantity, including audit fields for tracking changes.

      7.2.2. ZI_EMISSIONREPORT_UI : UI annotation for **ZEM_EMISSION** showing usage ID, source details, CO2 factor, and calculated total emissions.

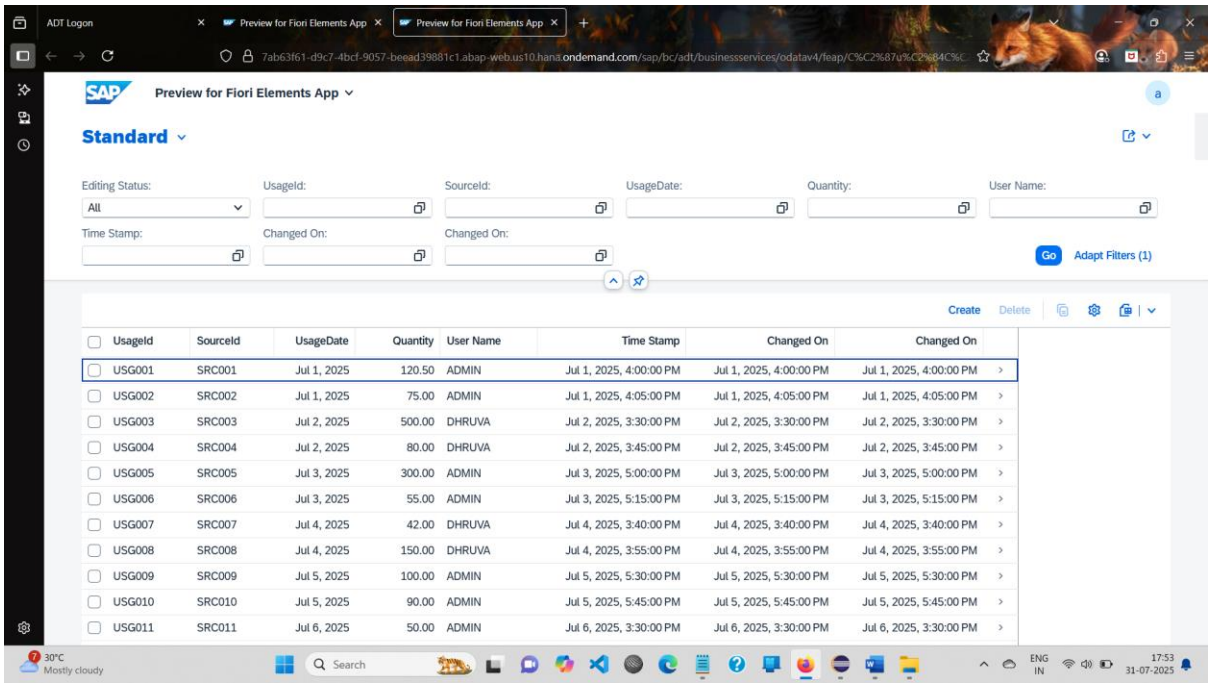
      7.2.3. ZI_CERTIFICATE_UI : UI annotation for **ZI_ELIGCERTIFICATE**, displaying certificate ID, emissions, credit score, and certificate status with issue and expiry dates.

8. **Service Definition & Binding**
   8.1. Service Definitions
   8.2. Service Bindings
      8.2.1. USAGE_LOG_TABLE
      8.2.2. EMISSION REPORT
      8.2.3. CERTIFICATE SUMMARY DASHBOARD

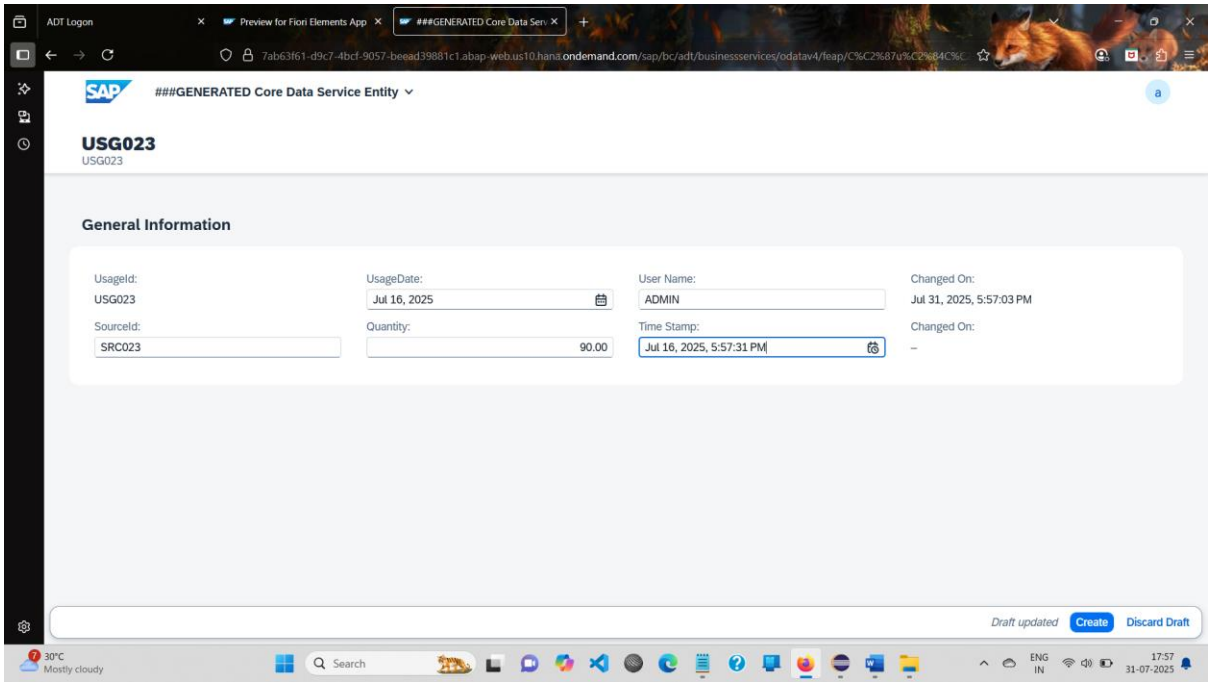
9. **Behaviour Implementation Classes**
   9.1. ZBP_CFM_USAGE_ENTITY – Handles insert operations for source, factor, and usage logs in the Carbon Emission Management system.

   9.2. ZBP_CERTIFY – Generates and inserts carbon emission certificates from credit reports with UUID-based IDs, setting status and expiry logic.

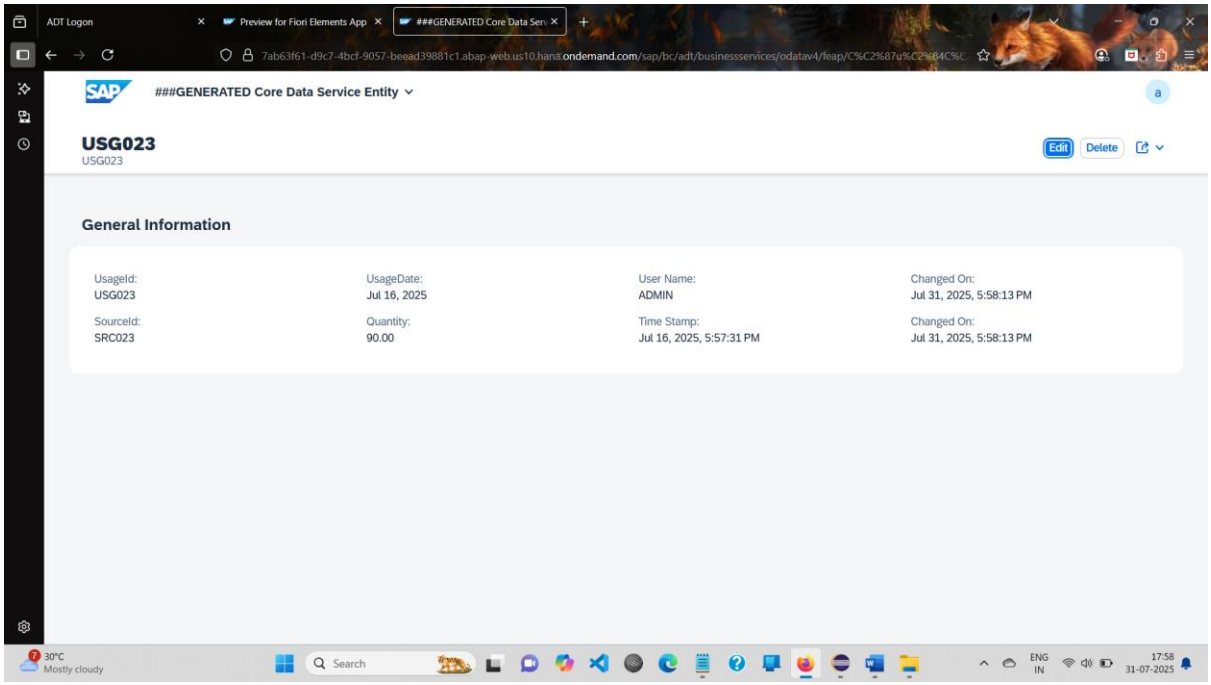
Carbon Usage Tracker



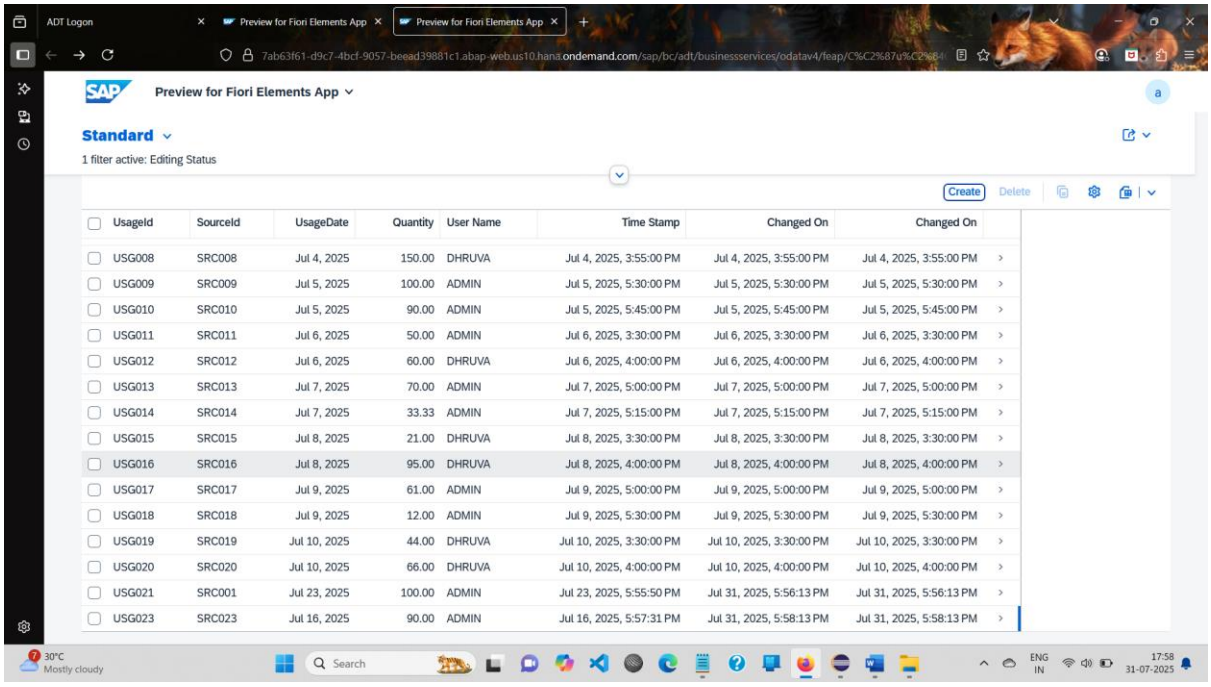
Create : -



Edit :-

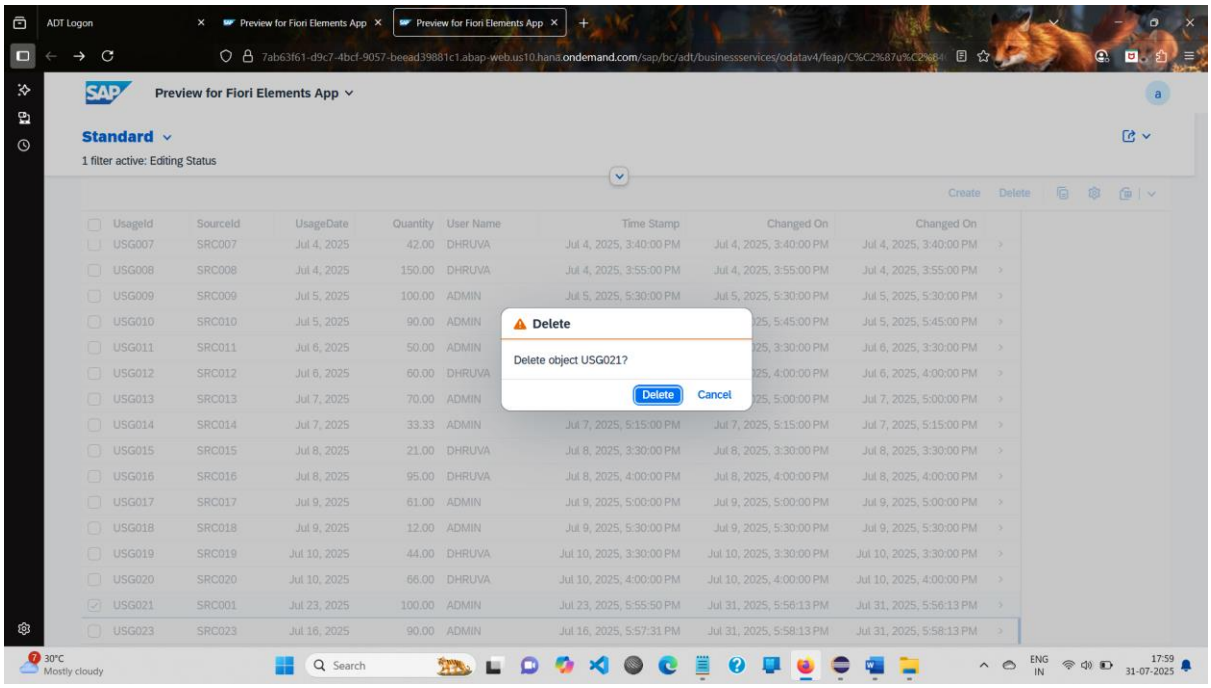


Read :-

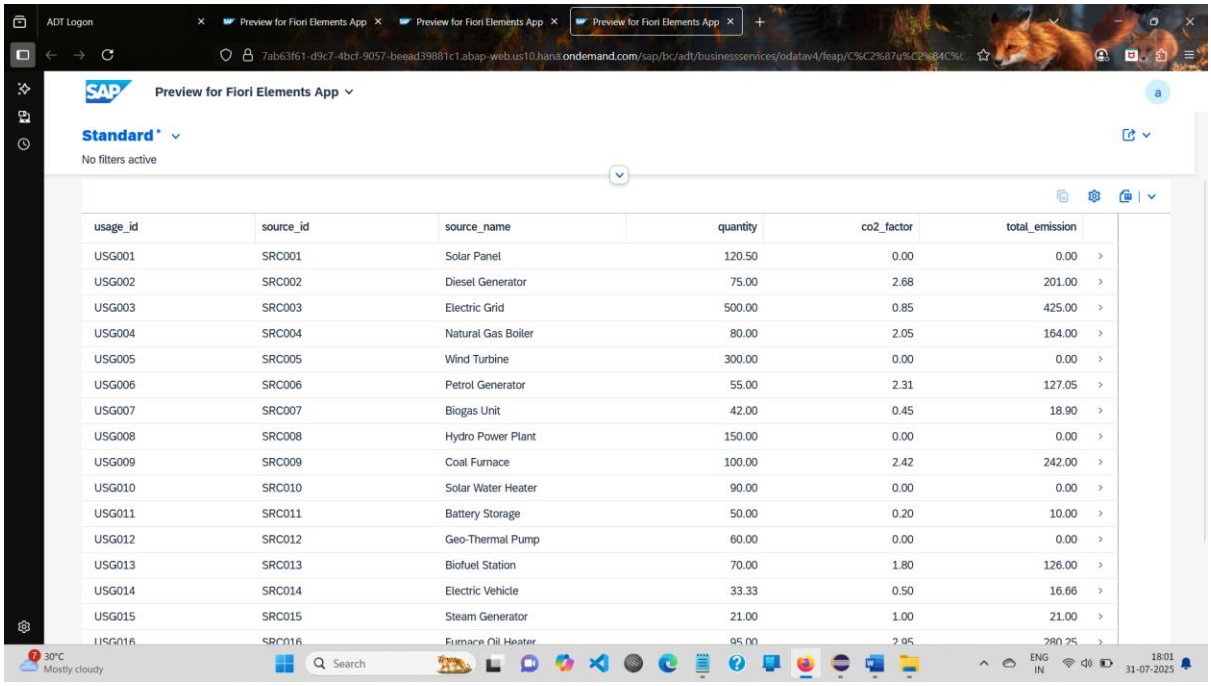


UsageId	SourceId	UsageDate	Quantity	User Name	Time Stamp	Changed On	Changed On
USG008	SRC008	Jul 4, 2025	150.00	DHRUVA	Jul 4, 2025, 3:55:00 PM	Jul 4, 2025, 3:55:00 PM	Jul 4, 2025, 3:55:00 PM
USG009	SRC009	Jul 5, 2025	100.00	ADMIN	Jul 5, 2025, 5:30:00 PM	Jul 5, 2025, 5:30:00 PM	Jul 5, 2025, 5:30:00 PM
USG010	SRC010	Jul 5, 2025	90.00	ADMIN	Jul 5, 2025, 5:45:00 PM	Jul 5, 2025, 5:45:00 PM	Jul 5, 2025, 5:45:00 PM
USG011	SRC011	Jul 6, 2025	50.00	ADMIN	Jul 6, 2025, 3:30:00 PM	Jul 6, 2025, 3:30:00 PM	Jul 6, 2025, 3:30:00 PM
USG012	SRC012	Jul 6, 2025	60.00	DHRUVA	Jul 6, 2025, 4:00:00 PM	Jul 6, 2025, 4:00:00 PM	Jul 6, 2025, 4:00:00 PM
USG013	SRC013	Jul 7, 2025	70.00	ADMIN	Jul 7, 2025, 5:00:00 PM	Jul 7, 2025, 5:00:00 PM	Jul 7, 2025, 5:00:00 PM
USG014	SRC014	Jul 7, 2025	33.33	ADMIN	Jul 7, 2025, 5:15:00 PM	Jul 7, 2025, 5:15:00 PM	Jul 7, 2025, 5:15:00 PM
USG015	SRC015	Jul 8, 2025	21.00	DHRUVA	Jul 8, 2025, 3:30:00 PM	Jul 8, 2025, 3:30:00 PM	Jul 8, 2025, 3:30:00 PM
USG016	SRC016	Jul 8, 2025	95.00	DHRUVA	Jul 8, 2025, 4:00:00 PM	Jul 8, 2025, 4:00:00 PM	Jul 8, 2025, 4:00:00 PM
USG017	SRC017	Jul 9, 2025	61.00	ADMIN	Jul 9, 2025, 5:00:00 PM	Jul 9, 2025, 5:00:00 PM	Jul 9, 2025, 5:00:00 PM
USG018	SRC018	Jul 9, 2025	12.00	ADMIN	Jul 9, 2025, 5:30:00 PM	Jul 9, 2025, 5:30:00 PM	Jul 9, 2025, 5:30:00 PM
USG019	SRC019	Jul 10, 2025	44.00	DHRUVA	Jul 10, 2025, 3:30:00 PM	Jul 10, 2025, 3:30:00 PM	Jul 10, 2025, 3:30:00 PM
USG020	SRC020	Jul 10, 2025	66.00	DHRUVA	Jul 10, 2025, 4:00:00 PM	Jul 10, 2025, 4:00:00 PM	Jul 10, 2025, 4:00:00 PM
USG021	SRC001	Jul 23, 2025	100.00	ADMIN	Jul 23, 2025, 5:55:50 PM	Jul 31, 2025, 5:56:13 PM	Jul 31, 2025, 5:56:13 PM
USG023	SRC023	Jul 16, 2025	90.00	ADMIN	Jul 16, 2025, 5:57:31 PM	Jul 31, 2025, 5:58:13 PM	Jul 31, 2025, 5:58:13 PM

Delete : -

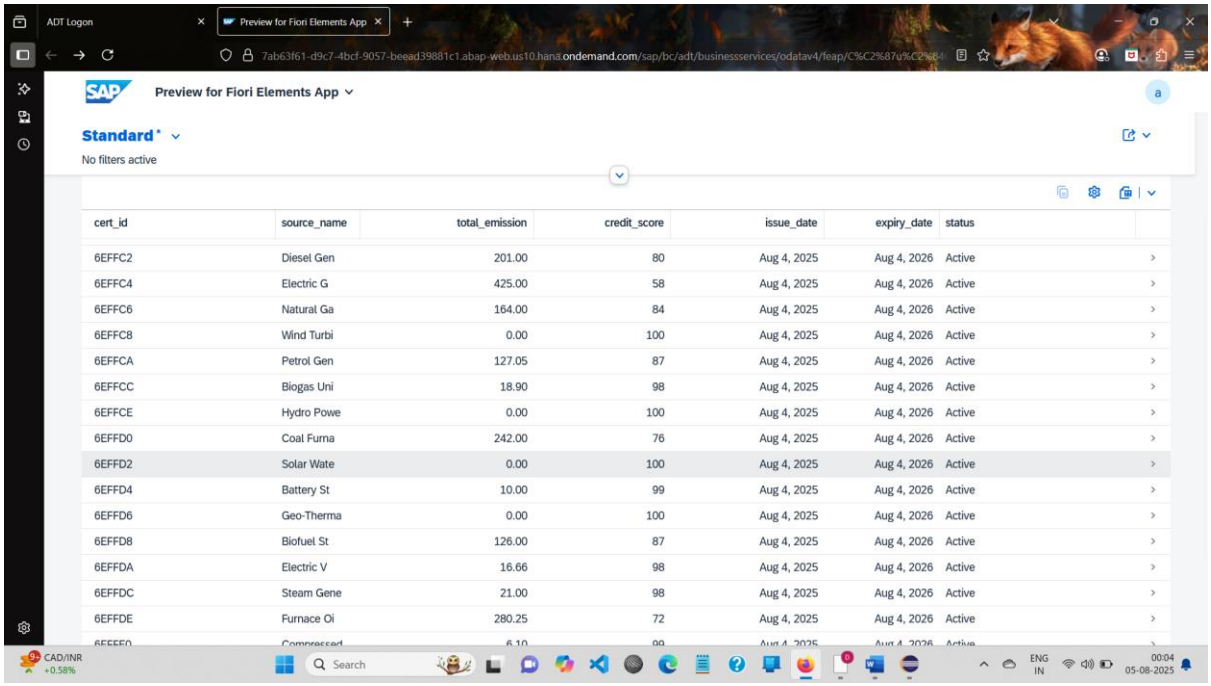


Emission Report



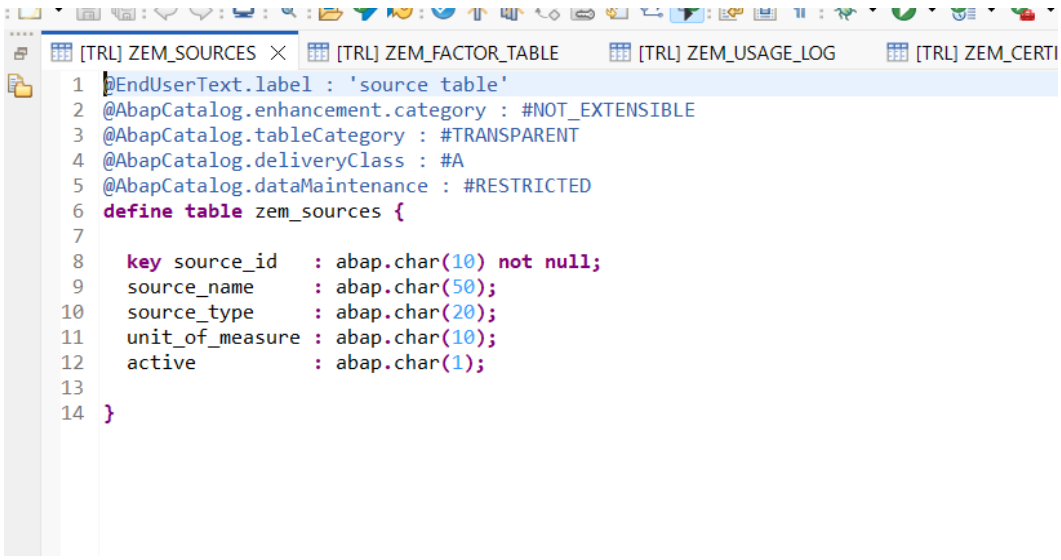
usage_id	source_id	source_name	quantity	co2_factor	total_emission
USG001	SRC001	Solar Panel	120.50	0.00	0.00
USG002	SRC002	Diesel Generator	75.00	2.68	201.00
USG003	SRC003	Electric Grid	500.00	0.85	425.00
USG004	SRC004	Natural Gas Boiler	80.00	2.05	164.00
USG005	SRC005	Wind Turbine	300.00	0.00	0.00
USG006	SRC006	Petrol Generator	55.00	2.31	127.05
USG007	SRC007	Biogas Unit	42.00	0.45	18.90
USG008	SRC008	Hydro Power Plant	150.00	0.00	0.00
USG009	SRC009	Coal Furnace	100.00	2.42	242.00
USG010	SRC010	Solar Water Heater	90.00	0.00	0.00
USG011	SRC011	Battery Storage	50.00	0.20	10.00
USG012	SRC012	Geo-Thermal Pump	60.00	0.00	0.00
USG013	SRC013	Biofuel Station	70.00	1.80	126.00
USG014	SRC014	Electric Vehicle	33.33	0.50	16.66
USG015	SRC015	Steam Generator	21.00	1.00	21.00
USG016	SRC016	Furnace Oil Heater	95.00	2.95	280.25

Certificate Summary Dashboard

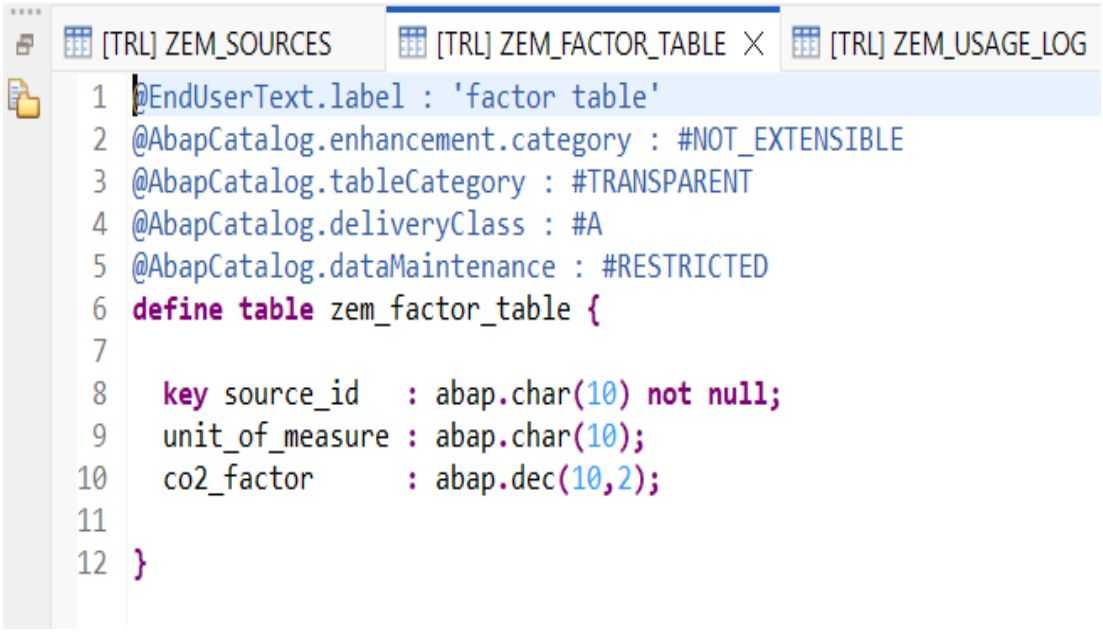


Database Tables :-

1. ZEM_SOURCES – Source Table



2. ZEM_FACTOR_TABLE – Factor Table



3. ZEM_USAGE_LOG – Usage Log Table

[TRL] ZEM_SOURCES | [TRL] ZEM_FACTOR_TABLE | [TRL] ZEM_USAGE_LOG | [TRL

```
@EndUserText.label : 'log table'
@AbapCatalog.enhancement.category : #NOT_EXTENSIBLE
@AbapCatalog.tableCategory : #TRANSPARENT
@AbapCatalog.deliveryClass : #A
@AbapCatalog.dataMaintenance : #RESTRICTED
define table zem_usage_log {

  key client        : abap.clnt not null;
  key usage_id      : abap.char(10) not null;
  source_id         : abap.char(10);
  usage_date        : abap.dats;
  quantity          : abap.dec(10,2);
  created_by        : syuname;
  created_at        : timestampl;
  loal_last_changed : abp_locinst_lastchange_tstmpl;
  last_changed      : abp_lastchange_tstmpl;

}
```

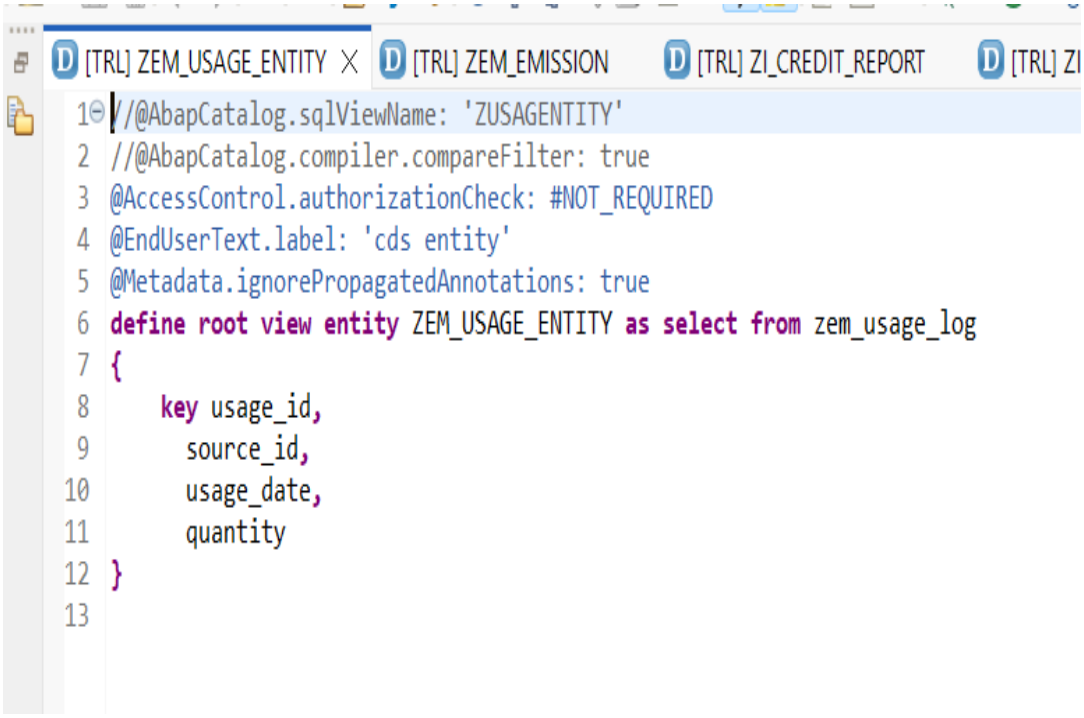
4. ZEM_CERTIFICATE – Certificate Table

[TRL] ZEM_CERTIFICATE | [TRL] ZBP_CERTIFY | [TRL] ZCL_CERTIFICATE

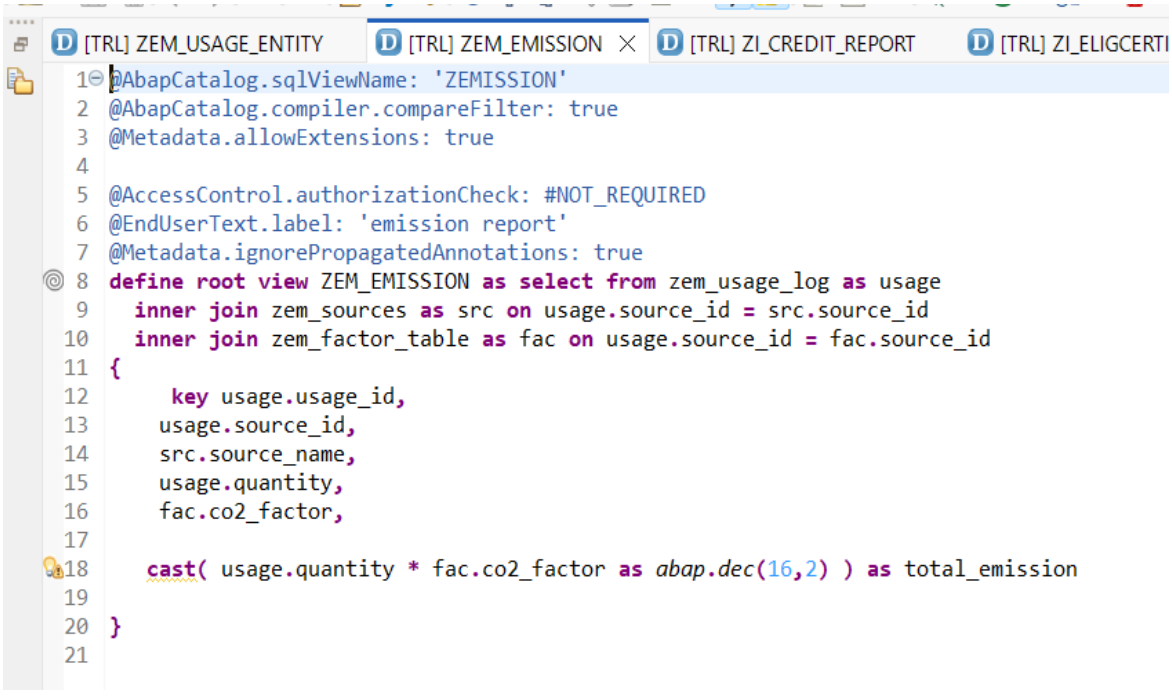
```
@EndUserText.label : 'certificate manage'
@AbapCatalog.enhancement.category : #NOT_EXTENSIBLE
@AbapCatalog.tableCategory : #TRANSPARENT
@AbapCatalog.deliveryClass : #A
@AbapCatalog.dataMaintenance : #RESTRICTED
define table zem_certificate {

  key client      : abap.clnt not null;
  key cert_id     : abap.char(32) not null;
  usage_id        : abap.char(10);
  source_name     : abap.char(10);
  total_emission  : abap.dec(10,2);
  credit_score    : abap.int4;
  issue_date      : abap.dats;
  expiry_date     : abap.dats;
  status          : abap.char(10);

}
```

7. CDS VIEWS (DATA DEFINITION) :-

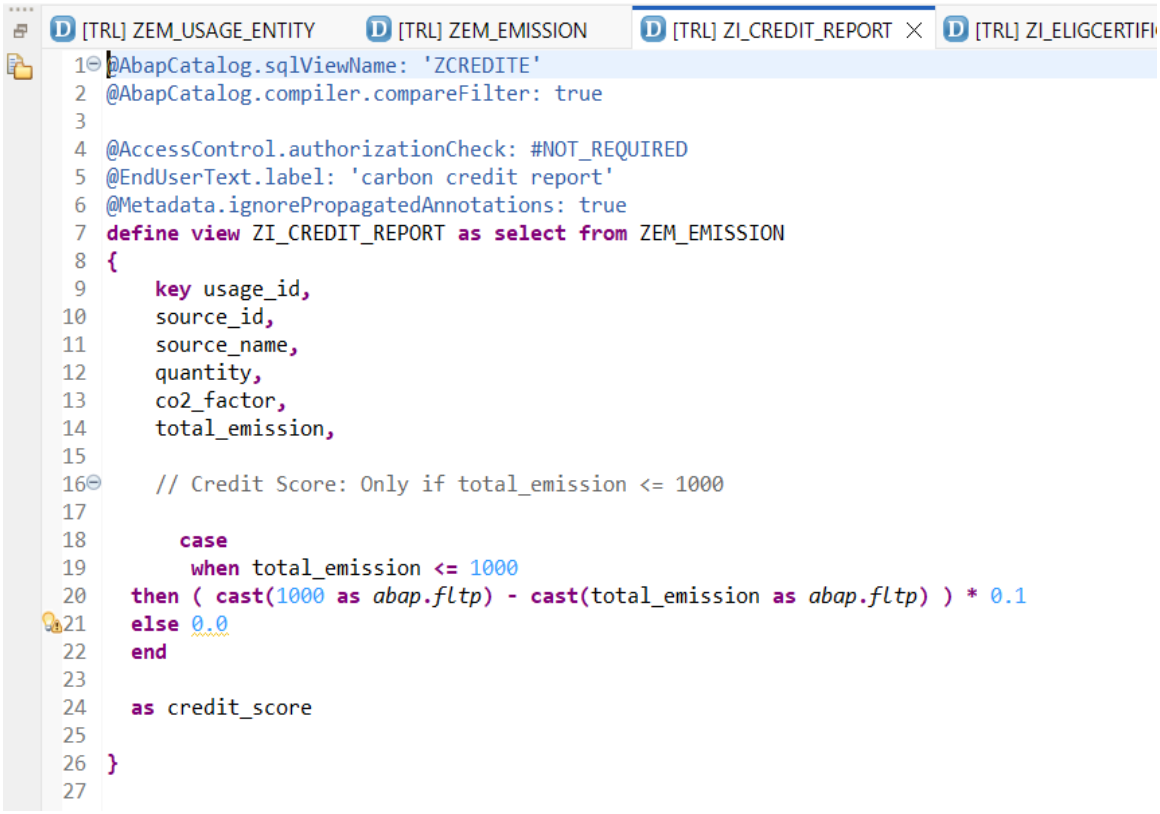
7.1.1 ZEM_USAGE_ENTITY



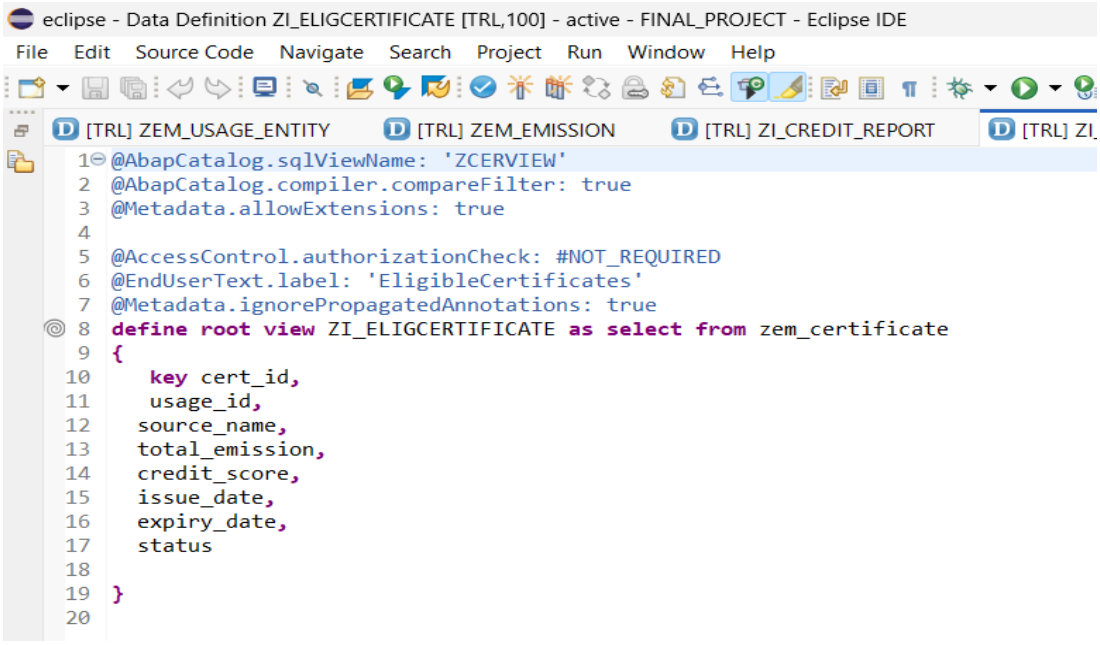
7.1.2 ZEM_EMISSION



7.1.3 ZI_CREDIT_REPORT



7.1.4 ZI_ELIGCERTIFICATE



Metadata Extension :-

7.2.1 ZC_EM_USAGE_LOG000

[TRL] ZC_EM_USAGE_LOG000 | [TRL] ZI_EMISSIONREPORT_UI | [TRL] ZI_CER

```
@Metadata.layer: #CORE
@UI.headerInfo.title.type: #STANDARD
@UI.headerInfo.title.value: 'UsageId'
@UI.headerInfo.description.type: #STANDARD
@UI.headerInfo.description.value: 'UsageId'
annotate view ZC_EM_USAGE_LOG000 with
{
  @EndUserText.label: 'UsageId'
  UsageId;

  @EndUserText.label: 'SourceId'
  SourceId;

  @EndUserText.label: 'UsageDate'
  UsageDate;

  @EndUserText.label: 'Quantity'
  Quantity;

  @UI.identification: [ {
  CreatedBy;

  @UI.identification: [ {
  CreatedAt;

  @UI.identification: [ {
  LoalLastChanged;

  @UI.identification: [ {
  LastChanged;
}
```

7.2.1 ZI_EMISSIONREPORT_UI

[TRL] ZC_EM_USAGE_LOG000 | [TRL] ZI_EMISSIONREPORT_UI

```
@Metadata.layer: #CORE
annotate entity ZEM_EMISSION
    with {
    @UI.lineItem: [{ position: 10 }]
  usage_id;

  @UI.lineItem: [{ position: 20 }]
  source_id;

  @UI.lineItem: [{ position: 30 }]
  source_name;

  @UI.lineItem: [{ position: 40 }]
  quantity;

  @UI.lineItem: [{ position: 50 }]
  co2_factor;

  @UI.lineItem: [{ position: 60 }]
  total_emission;



}
```

7.2.3 ZI_CERTIFICATE_UI

[TRL] ZC_EM_USAGE_LOG000 [TRL] ZI_EMISSIONREPORT_UI

```
@Metadata.layer: #CORE
annotate entity ZI_ELIGCERTIFICATE
    with
{
    @UI.lineItem: [{ position: 10 }]
  cert_id;

  @UI.lineItem: [{ position: 20 }]
  source_name;

  @UI.lineItem: [{ position: 30 }]
  total_emission;

  @UI.lineItem: [{ position: 40 }]
    credit_score;

  @UI.lineItem: [{ position: 50 }]
    issue_date;

  @UI.lineItem: [{ position: 60 }]
    expiry_date;

  @UI.lineItem: [{ position: 70 }]
    status;



}
```

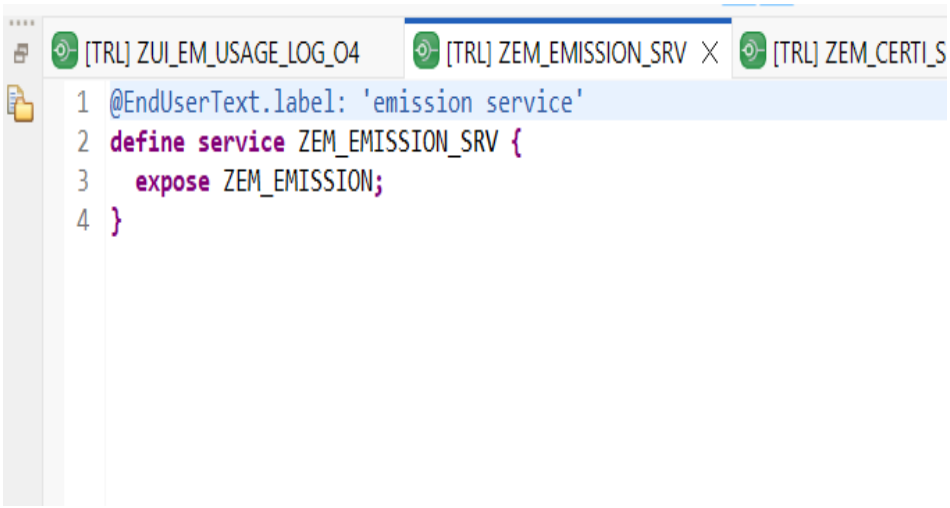
8. Service Definition :-

ZUI_EM_USAGE_LOG_04

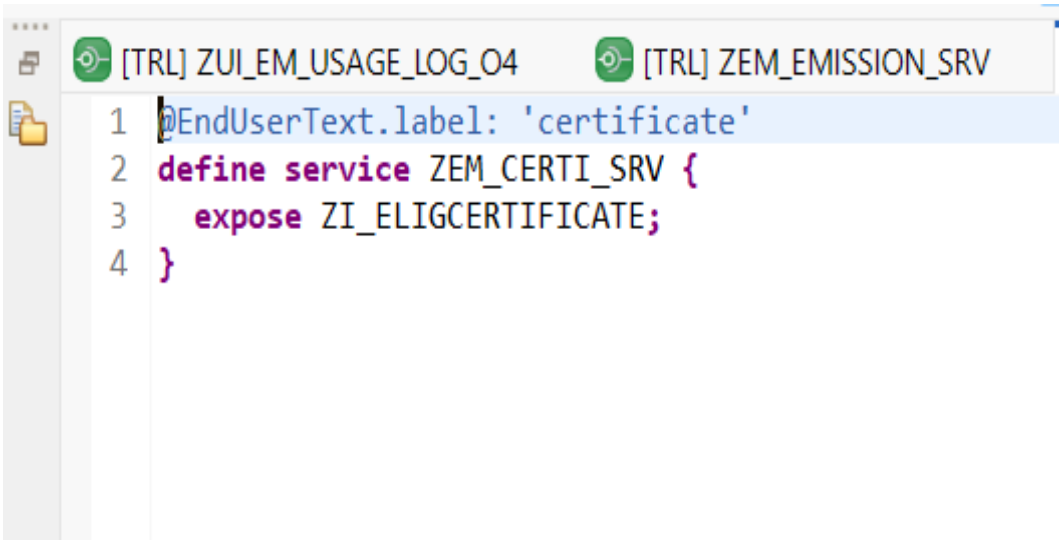
[TRL] ZUI_EM_USAGE_LOG_O4 [TRL] ZEM_EMISSION_SRV [TRL] ZEM_CERTI_SRV

```
@EndUserText: {
  label: 'Service Definition for ZC_EM_USAGE_LOG000'
}
@ObjectModel: {
  leadingEntity: {
    name: 'ZC_EM_USAGE_LOG000'
  }
}
define service ZUI_EM_USAGE_LOG_O4 provider contracts odata_v4_ui {
  expose ZC_EM_USAGE_LOG000 as carbon;
}
```

ZEM_EMISSION_SRV

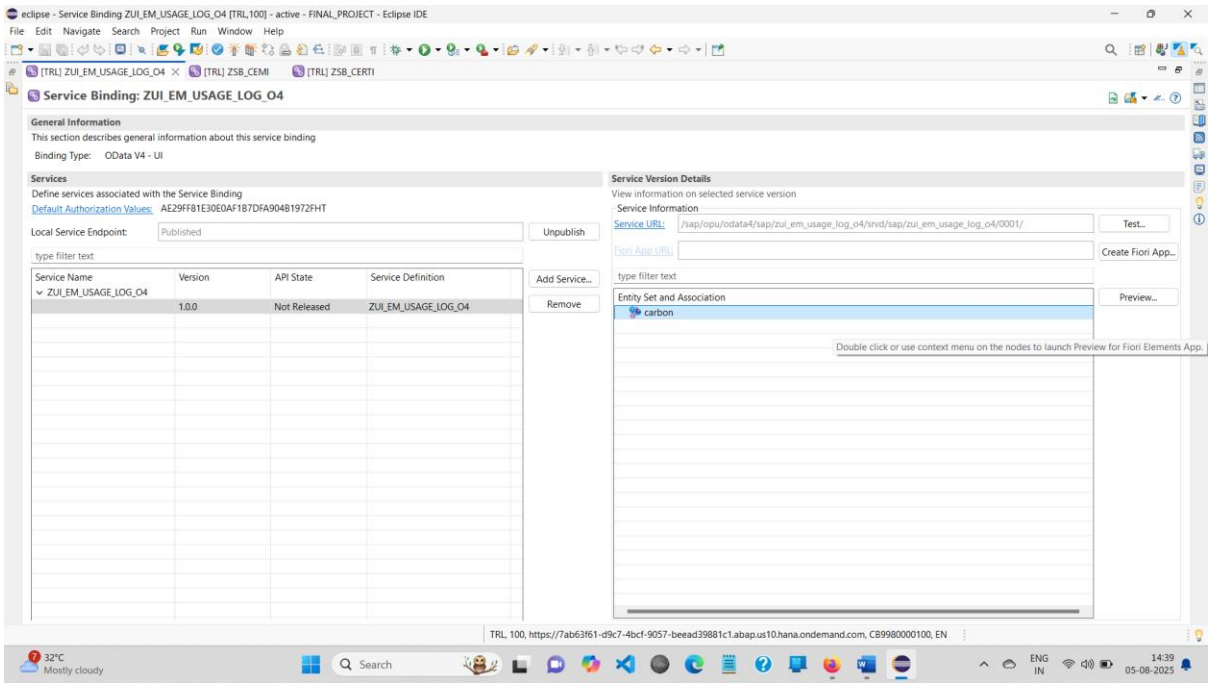


ZEM_CERTI_SRV

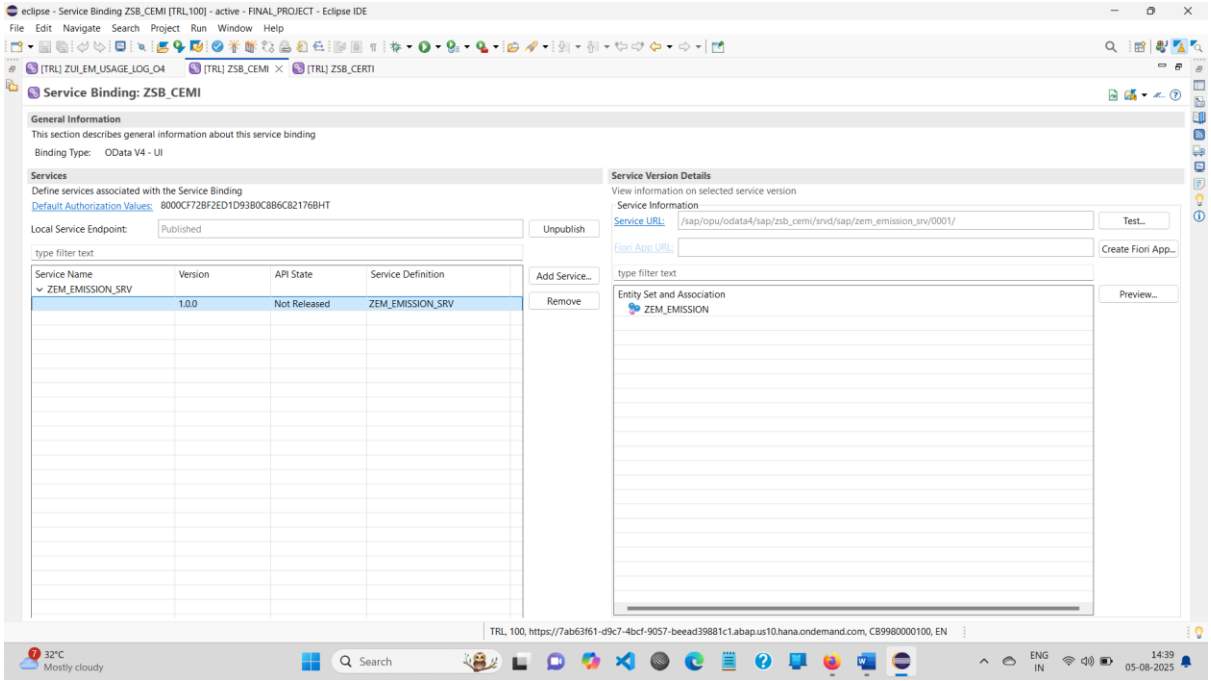


Service Binding :-

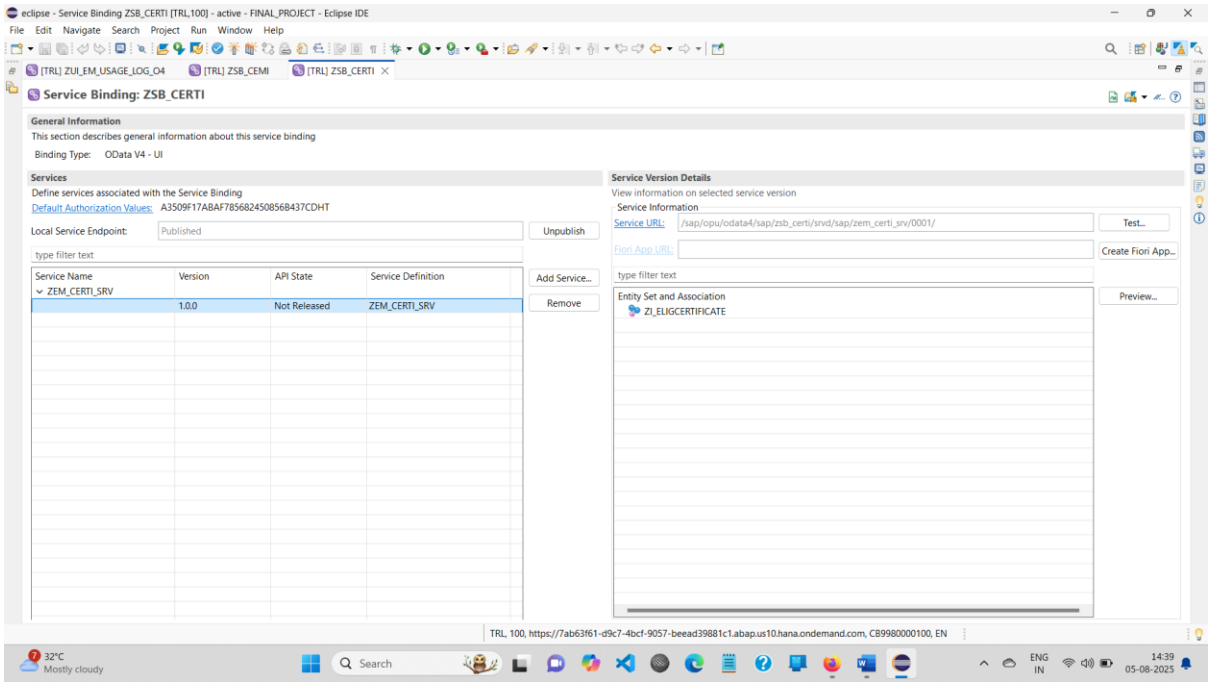
USAGE_LOG_TABLE



EMISSION_REPORT



CERTIFICATE_DASHBOARD



9. Behaviour Implementation Classes :-

CLASS : ZBP_CEM_USAGE_ENTITY

```
CLASS zbp_cem_usage_entity DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC .

  PUBLIC SECTION.
   METHODS:
     insert_source      IMPORTING is_source TYPE zem_sources
                  RETURNING VALUE(rv_message) TYPE string,
     insert_factor     IMPORTING is_factor TYPE zem_factor_table
                   RETURNING VALUE(rv_message) TYPE string,

     insert_usage_log  IMPORTING is_usage TYPE zem_usage_log
                  RETURNING VALUE(rv_message) TYPE string.
   INTERFACES if_oo_adt_classrun .
  PROTECTED SECTION.
  PRIVATE SECTION.
```

```
ENDCLASS.

CLASS zbp_cem_usage_entity IMPLEMENTATION.

  METHOD if_oo_adt_classrun~main.

    DATA lt_sources TYPE STANDARD TABLE OF zem_sources.
  DATA lv_message TYPE string.

  " Add multiple source entries
  lt_sources = VALUE #(
    ( source_id = 'SRC001' source_name = 'Solar Panel'            source_type = 'Renewable'
unit_of_measure = 'kWh'  active = 'X' )
  ( source_id = 'SRC002' source_name = 'Diesel Generator'       source_type = 'Fuel'
unit_of_measure = 'Litre' active = 'X' )
  ( source_id = 'SRC003' source_name = 'Electric Grid'          source_type = 'Power'
unit_of_measure = 'kWh'  active = 'X' )
  ( source_id = 'SRC004' source_name = 'Natural Gas Boiler'     source_type = 'Fuel'
unit_of_measure = 'm3'   active = 'X' )
  ( source_id = 'SRC005' source_name = 'Wind Turbine'           source_type = 'Renewable'
unit_of_measure = 'kWh'  active = 'X' )

  ).

  " Loop and insert each one
  LOOP AT lt_sources INTO DATA(ls_source).
    lv_message = insert_source( ls_source ).
    out->write( lv_message ).
  ENDLOOP.

  DATA: lt_factors TYPE STANDARD TABLE OF zem_factor_table,
      ls_factor  TYPE zem_factor_table.

lt_factors = VALUE #(
  ( source_id = 'SRC001' unit_of_measure = 'kWh'    co2_factor = '0.00' )  " Solar Panel
  ( source_id = 'SRC002' unit_of_measure = 'Litre'  co2_factor = '2.68' )  " Diesel
  ( source_id = 'SRC003' unit_of_measure = 'kWh'    co2_factor = '0.85')  " Electric Grid
  ( source_id = 'SRC004' unit_of_measure = 'm3'     co2_factor = '2.05' )  " Natural Gas
  ( source_id = 'SRC005' unit_of_measure = 'kWh'    co2_factor = '0.00' )  " Wind

).

LOOP AT lt_factors INTO ls_factor.
  lv_message = insert_factor( ls_factor ).  " Call your insert method
  out->write( lv_message ).
ENDLOOP.

    DATA: lt_usages TYPE STANDARD TABLE OF zem_usage_log,
      ls_usage  TYPE zem_usage_log.
      DATA(lv_today) = cl_abap_context_info=>get_system_date( ).

lt_usages = VALUE #(
  ( client = '100' usage_id = 'USG001' source_id = 'SRC001' usage_date = '20250701' quantity
= '120.50' created_by = 'ADMIN' created_at = '20250701103000' loal_last_changed =
'20250701103000' last_changed = '20250701103000' )
```

```
  ( client = '100' usage_id = 'USG002' source_id = 'SRC002' usage_date = '20250701' quantity
= '75.00'  created_by = 'ADMIN' created_at = '20250701103500' loal_last_changed =
'20250701103500' last_changed = '20250701103500' )
  ( client = '100' usage_id = 'USG003' source_id = 'SRC003' usage_date = '20250702' quantity
= '500.00' created_by = 'DHRUVA' created_at = '20250702100000' loal_last_changed =
'20250702100000' last_changed = '20250702100000' )
  ( client = '100' usage_id = 'USG004' source_id = 'SRC004' usage_date = '20250702' quantity
= '80.00'  created_by = 'DHRUVA' created_at = '20250702101500' loal_last_changed =
'20250702101500' last_changed = '20250702101500' )
  ( client = '100' usage_id = 'USG005' source_id = 'SRC005' usage_date = '20250703' quantity
= '300.00' created_by = 'ADMIN' created_at = '20250703113000' loal_last_changed =
'20250703113000' last_changed = '20250703113000' )
).

LOOP AT lt_usages INTO ls_usage.
  lv_message = insert_usage_log( ls_usage ).  " Call your insert method
  out->write( lv_message ).
ENDLOOP.

  ENDMETHOD.

  METHOD insert_source.
    " Basic check for existing source
    DATA: lv_exists TYPE zem_sources-source_id.

    SELECT SINGLE source_id
      FROM zem_sources
      WHERE source_id = @is_source-source_id
      INTO @lv_exists.

    IF sy-subrc = 0.
      rv_message = |Source ID already exists: { is_source-source_id }| .
      RETURN.
    ENDIF.

    INSERT zem_sources FROM @is_source.
    IF sy-subrc = 0.
      rv_message = |Source inserted: { is_source-source_id }| .
    ELSE.
      rv_message = |Failed to insert source: { is_source-source_id }|.
    ENDIF.
  ENDMETHOD.

  METHOD insert_factor.
    " Optional check: Ensure source exists
    SELECT SINGLE source_id
      FROM zem_sources
      WHERE source_id = @is_factor-source_id
     INTO @DATA(lv_source).
    IF sy-subrc <> 0.
      rv_message = |Invalid source ID for factor: { is_factor-source_id }| .
      RETURN.
    ENDIF.

    INSERT zem_factor_table FROM @is_factor.
```

```
    IF sy-subrc = 0.
      rv_message = |Factor inserted for source: { is_factor-source_id }| .
    ELSE.
      rv_message = |Failed to insert factor for source: { is_factor-source_id }| .
    ENDIF.
  ENDMETHOD.

  METHOD insert_usage_log.
    " Optional: Validate source exists
    SELECT SINGLE source_id
      FROM zem_sources
      WHERE source_id = @is_usage-source_id
      INTO @DATA(lv_valid_source).
    IF sy-subrc <> 0.
      rv_message = |Invalid source ID in usage log: { is_usage-source_id }| .
      RETURN.
    ENDIF.

    " Validate: Quantity must be greater than 0
    IF is_usage-quantity <= 0.
      rv_message = |Quantity must be greater than 0 for usage ID: { is_usage-usage_id }| .
      RETURN.
    ENDIF.

    INSERT zem_usage_log FROM @is_usage.
    IF sy-subrc = 0.
     rv_message = |Usage log inserted: { is_usage-usage_id }| .
    ELSE.
      rv_message = |Failed to insert usage log: { is_usage-usage_id }| .
    ENDIF.
  ENDMETHOD.

ENDCLASS.
```

CLASS : ZBP_CERTIFY

```
CLASS zbp_certify DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC .

  PUBLIC SECTION.

    INTERFACES if_oo_adt_classrun .
  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.

CLASS zbp_certify IMPLEMENTATION.

  METHOD if_oo_adt_classrun~main.
  DATA: lt_report TYPE TABLE OF zi_credit_report,
        ls_cert   TYPE zem_certificate,
```

```
        lv_count  TYPE i VALUE 0,
        lv_uuid_string TYPE string,
        lv_uuid_full   TYPE string,
      lv_short_uuid  TYPE string.

    " 1. Fetch data from CDS View
    SELECT * FROM zi_credit_report INTO TABLE @lt_report.

    " 2. Loop through each and insert into ZEM_CERTIFICATE
    LOOP AT lt_report INTO DATA(ls_report).

      CLEAR : ls_cert,lv_uuid_string.

      " Populate certificate structure
     TRY.
        lv_uuid_full = cl_system_uuid=>if_system_uuid_static~create_uuid_c32( ).
        lv_short_uuid = lv_uuid_full+0(3) && lv_uuid_full+26(3).

        ls_cert-cert_id = lv_short_uuid. " Use full UUID (char32)


       CATCH cx_uuid_error INTO DATA(lx_uuid).
         out->write( |⚠ UUID generation failed: { lx_uuid->get_text( ) }| ).
         CONTINUE.
      ENDTRY.
      ls_cert-usage_id       = ls_report-usage_id.
      ls_cert-source_name    = ls_report-source_name.
      ls_cert-total_emission = ls_report-total_emission.
      ls_cert-credit_score   = ls_report-credit_score.
      ls_cert-issue_date     = cl_abap_context_info=>get_system_date( ).
      ls_cert-expiry_date = cl_abap_context_info=>get_system_date( ) + 365.
      IF ls_cert-expiry_date < cl_abap_context_info=>get_system_date( ) .
    ls_cert-status = 'Expired'.
   ELSE.
    ls_cert-status = 'Active'.
   ENDIF.

      " Insert into database table
      INSERT zem_certificate FROM @ls_cert.
      IF sy-subrc = 0.
        lv_count = lv_count + 1.
      ENDIF.

    ENDLOOP.

    out->write( |Inserted { lv_count } certificate(s) into ZEM_CERTIFICATE.| ).

  ENDMETHOD.
ENDCLASS.
```

- ## Final Outcome

  This RAP-based carbon certification system enables automated credit score assignment, certificate generation, and expiry validation using CDS views, ABAP classes, and behavior definitions. It ensures clean-core architecture and integrates seamlessly with Fiori Elements for a user-friendly dashboard and real-time tracking of emission activities.

- ## Conclusion

  The Carbon Certificate Management System, built using the ABAP RESTful Application Programming Model (RAP), delivers a scalable, maintainable, and cloud-ready solution for managing carbon emissions and credit certification processes. Leveraging CDS views, behavior definitions, and service binding with OData V4, the system ensures seamless backend integration and a clean-core architecture aligned with SAP's modern development principles.

  Key functionalities such as emission tracking, credit score calculation, certificate generation, expiry validation, and duplicate prevention are efficiently handled through encapsulated ABAP logic and behavior implementation. The system also features a real-time summary dashboard using Fiori Elements, enabling organizations to monitor emission trends and take proactive sustainability actions.

  With its modular design and enterprise-grade extensibility, this RAP-based solution offers a future-proof framework for environmental compliance and corporate responsibility initiatives. It meets industry needs while maintaining compatibility with SAP BTP and Fiori UX standards.