UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

COMPUTER SCIENCE DEPARTMENT

Computer Science - Curriculum Artificial Intelligence

Project Assignment

Database Systems

Database Design and Implementation

Student: *Fontana Emanuele*

Academic Year 2024/2025

Indice

1 Conceptual Design

1.1 Requirements

The requirements for the database are the following:

"GreenWorld Energy"
The company "GreenWorld Energy" operates decentralized renewable energy production facilities distributed across regions. Each facility is characterized by a name, location, type of energy produced (e.g., solar, wind, hydro), and maximum energy output capacity. The company also manages contracts with customers for energy supply, categorized as residential or commercial, and offers flexible pricing models based on consumption. Each customer has one or more accounts, each identified by a unique code. Every energy contract is linked to a single customer account and includes details such as start date, duration, energy plan, and cost. Facilities are overseen by management teams, each identified by a unique code, team name, and the number of projects managed. Teams are evaluated based on performance metrics such as energy efficiency, uptime, and customer satisfaction. Each team is represented by the main responsible employee and other employees identified by fiscal code, name, surname, date of birth and date of hiring. Additionally, the company supports a feedback system allowing customers to submit ratings and comments regarding service quality. Customers are classified into residential and commercial types, each identified by a unique alphanumeric code, with associated contact details and energy consumption history.

Table 1: Requirements

1.2 Analysis

1.2.1 Glossary of Terms

Term	Description	Connections	Synonyms
Region	Area where the facilities are located.	Facility	Location
Facility	Energy production facility. It emits energy for customers	Contract,Teams,Region	Projects
Contract	Energy supply contract. It describes the energy plan. It can be residential or commercial	Account, Facility	
Customer	Customer of GreenWorld Energy. It can be residential or commercial	Account	
Account	Customer account. It associated with only one account and one contract	Contract, Customer, Feedback	
Feedback	Customer feedback. Used to give a score to each team	Account,Team	Ratings, customer satisfaction
Team	Groups of employees that oversees a facility. Each team has a manager	Facility, Employee, Feedback	
Employee	Employee of a GreenWorld Energy's team.	Team	

Table 2: Glossary of Terms

1.2.2 Level of Abstraction

By considering the synonyms of the terms, we can identify the level of abstraction of the terms. The terms can be classified as follows:

- **Region** as Location
- **Facility** as Facility
- **Contract** as Contract
- **Customer** as Customer
- **Account** as Account
- **Feedback** as Feedback
- **Team** as Team
- **Employee** as Employee

1.2.3 Reorganization of Concepts

The concepts can be reorganized as follows:

Facility
Each facility is characterized by a name, location, type of energy produced (e.g., solar, wind, hydro), and maximum energy output capacity. The company also manages contracts with customers for energy supply, categorized as residential or commercial, and offers flexible pricing models based on consumption. Facilities are overseen by management teams

Table 3: Facility's Concepts

Contract
The company also manages contracts with customers for energy supply, categorized as residential or commercial, and offers flexible pricing models based on consumption. Every enegry contract is linked to a single customer account and includes details such as start date, duration, energy plan, and cost.

Table 4: Contract's Concepts

Customer
Each customer has one or more accounts, each identified by a unique code. Customers are classified into residential and commercial types, each identified by a unique alphanumeric code, with associated contact details and energy consumption history. Additionally, the company supports a feedback system allowing customers to submit ratings and comments regarding service quality.

Table 5: Customer's Concepts

Account
Each customer has one or more accounts, each identified by a unique code. Every energy contract is linked to a single customer account and includes details such as start date, duration, energy plan, and cost.

Table 6: Account's Concepts

Feedback
The company supports a feedback system allowing customers to submit ratings and comments regarding service quality.

Table 7: Feedback's Concepts

Team
Facilities are overseen by management teams, each identified by a unique code, team name, and the number of projects managed. Teams are evaluated based on performance metrics such as energy efficiency, uptime, and customer satisfaction. Each team is represented by the main responsible employee

Table 8: Team's Concepts

Employee
Each team is represented by the main responsible employee and other employees identified by fiscal code, name, surname, date of birth and date of hiring.

Table 9: Employee's Concepts

1.2.4 Entity-Relationship Diagram

SKELETON SCHEMA

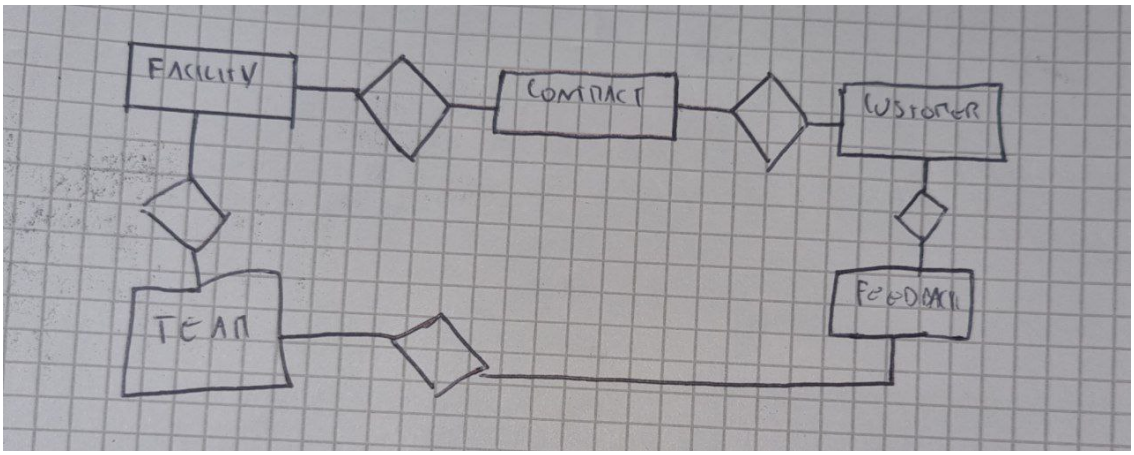


Figure 1: Skeleton Schema

This schema represents the main entities and their relationships. The entities are the following:

- **Facility**
- **Contract**
- **Customer**
- **Feedback**
- **Team**

The purpose of this schema is to have a general idea of the entities and their relationships. The attributes of the entities and the cardinality of the relationships are not included in this schema. Also, some entities are not included in this schema, such as Account and Employee. These entities will be included in the final schema with all the other details.

FINAL SCHEMA

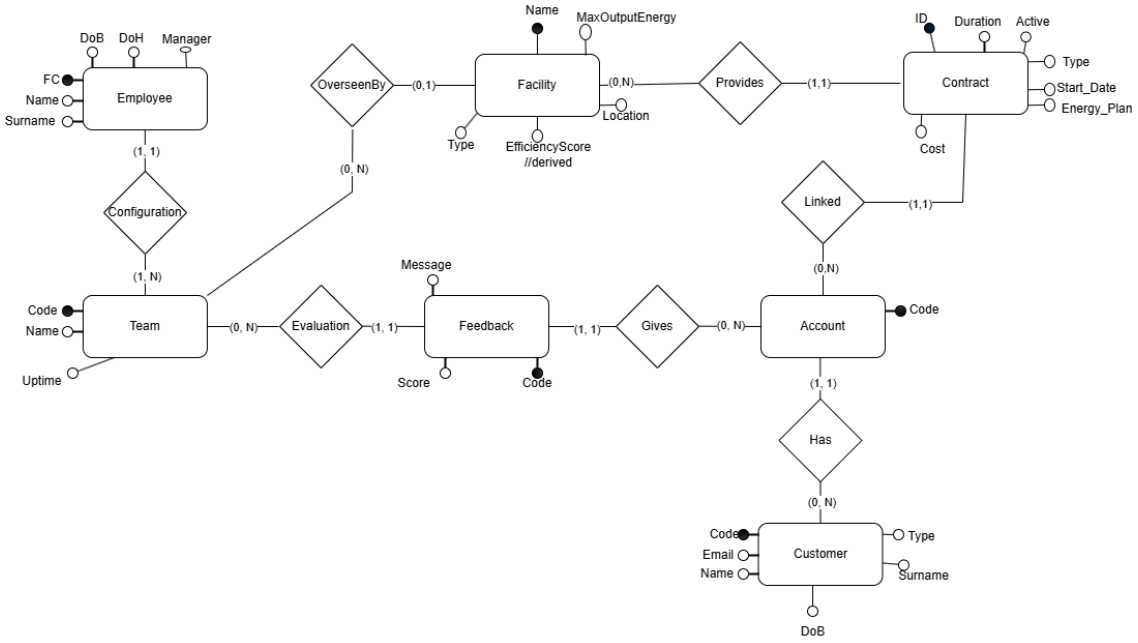


Figure 2: Final Schema

Note: Duration is expressed in months

1.2.5 Business Rules

The business rules are the following:

- A customer can't be a minor
- An employee can't be a minor
- Each team has got only one main responsible employee
- The sum of energy plan of all contracts related to a facility can't exceed the maximum energy output capacity of the facility
- If an account needs to activate a new contract with the same facility, the old contract must deactivated before the new one is activated
- A customer of a specific type can't have a contract of the other type
- The score of a feedback can be between 0 and 5
- Uptime is the number of hours the facility is active in a day. Between 6 and 8

- Energy efficiency is the percentage of energy produced compared to the maximum energy output capacity of the facility. Between 0 and 100
- EnergyPlan can be; 2.500 kWh, 4.500 kWh or 10.000 kWh
- Cost: 500 if EnergyPlan=2.500, 750 if EnergyPlan=4.500, 1000 if EnergyPlan=10.000

2 Logical Design

2.1 Volume Table

Lets consider the volumes for a year

Concept	Type	Volume	Description
Facility	E	100	Number of facilities (Given)
Contract	E	900.000	Number of contracts. 100 facilities * 500 contracts per facility * 12 months * 1.5 contracts per account
Customer	E	300.000	Number of customers.(ASSUMPTION)
Account	E	600.000	Number of accounts. 300.000 customers * 2 accounts per customer on average
Team	E	50	Number of teams. (ASSUMPTION)
Employee	E	250	Number of employees. 5 employees per team * 50 teams (ASSUMPTION)
Feedback	E	600.000	Number of feedbacks.
Provides	R	900.000	Each contract is provided by a single facility
Linked	R	900.000	On average each account has 1.5 contracts
Has	R	600.000	Each customer 2 accounts on average
OverseenBy	R	100	Each facility is overseen by a single team. Each team oversees 2 facilities, on average
Configuration	R	500	Each team constists of 5 employees on average
Gives	R	500.000	Each feedback is given by a single account.Not all accounts give feedback
Evaluation	R	500.000	Each feedback is linked to a single team

Table 10: Volume Table

2.2 Access Table

Operation1: Register a new customer (50 times per day)

Concept	Type	Access	Type
Customer	E	1	W

Table 11: Access Table for Operation1

Total Cost: 2 * 50 = 100 per day
Operation2: Register a new energy contract (50 times per day)

Concept	Type	Access	Type
Contract	E	1	W
Provides	R	1	W
Facility	E	1	R
Linked	R	1	W
Account	E	1	R
Facility	E	1	W
Provides	R	9000	R
Contract	E	9000	R

Table 12: Access Table for Operation2

Total Cost: (2+2+1+2+1+2+9000+9000)*200=3.602.000 per day
Every time we need to update the efficiency score of the facility related to the contracts.
Operation3: Assign a facility to a management team (50 times per day)

Concept	Type	Access	Type
Team	E	1	R
OverseenBy	R	1	W
Facility	E	1	R

Table 13: Access Table for Operation3

Total Cost: (1+2+1)*50=200 per day
Operation4: View the total energy output of a specific facility managed by the

eldest employee (1 per month = 0.03 per day)

Concept	**Type**	**Access**	**Type**
Employee	E	250	R
Configuration	R	1	R
Team	E	1	R
OverseenBy	R	2	R
Facility	E	2	R
Provides	R	9000	R
Contract	E	9000	R

Table 14: Access Table for Operation4

Total Cost: (250+1+1+1+2+9000+9000)*0.03=547.65 per day
Each team oversees, on average, twow facilities. We need to choose only one of them, which provides on average 9000 contracts. In this case we don't have sumEnergyOutput in the schema as attribute of the facility, so we need to compute it every time. If we introduce it, the cost will be

Concept	**Type**	**Access**	**Type**
Employee	E	250	R
Configuration	R	1	R
Team	E	1	R
OverseenBy	R	2	R
Facility	E	2	R

Table 15: Access Table for Operation4 with redundancy

Total Cost: (250+1+1+2+2)*0.03=7.68 per day
But in this case we need to update it every time we add a new contract, so the total cost of **Operation2** will be:

Concept	Type	Access	Type
Contract	E	1	W
Provides	R	1	W
Facility	E	1	R
Facility	E	1	W
Linked	R	1	W
Account	E	1	R
Facility	E	1	W
Provides	R	9000	R
Contract	E	9000	R

Table 16: Access Table for Operation2 with redundancy

Total Cost: (2+2+1+2+2+1+2+9000+9000)*200=3.602.400 per day
So the total cost without redundancy is 3.602.000 +547.65 = 3.602.547.65, while the total cost with redundancy is 3.602.400 + 7.68 = 3.602.407.68 per day. So we should keep the redundancy in this case.
Operation5: Print a ranked list of facilities based on their efficiency scores (10 per day)

Concept	Type	Access	Type
Facility	E	100	R

Table 17: Access Table for Operation5

Total Cost: 100*10 = 1000 per day
We can try to remove the redundancy of the efficiencyScore in facilities

Concept	Type	Access	Type
Facility	E	100	R
Provides	R	900.000	R
Contract	E	900.000	R

Table 18: Access Table for Operation5 without redundancy

Total Cost: (100+900.000+900.000)*10 = 18.001.000 per day

Concept	Type	Access	Type
Contract	E	1	W
Provides	R	1	W
Facility	E	1	R
Linked	R	1	W
Account	E	1	R

Table 19: Access Table for Operation2 without redundancy

Total Cost: (2+2+1+2+1)*200=1.600 per day
Now we don't need to update the the efficiency score everytime we register a new contract.
So the total cost without redundancy is 18.002.600, while the total cost with redundancy is 2.402.000 + 1000 = 2.403.000 per day.
We shouuld keep the redundancy in this case.

2.2.1 Redundancies and Generalization

- **Redundancies:** The number of facilities managed by a team is not stored in the schema, as it can be calculated by counting the number of facilities managed by the team. The score of the team can be derived by the following formula: $score = \frac{energy\_efficiency+uptime+AVG(customer\_score)}{3}$ and is not stored in the schema to avoid redundancy. Due to the previous analysis, the redundancy of the efficiency score and sumEngergyOutput in the facility are kept in the schema

- **Generalization:** The schema doesn't contain any generalization since there aren't different operations for them. There is a **type** attribute in the **Customer** and **Contract** entities so that the business rule related to the type of contract can be enforced.

2.3 UML Schema

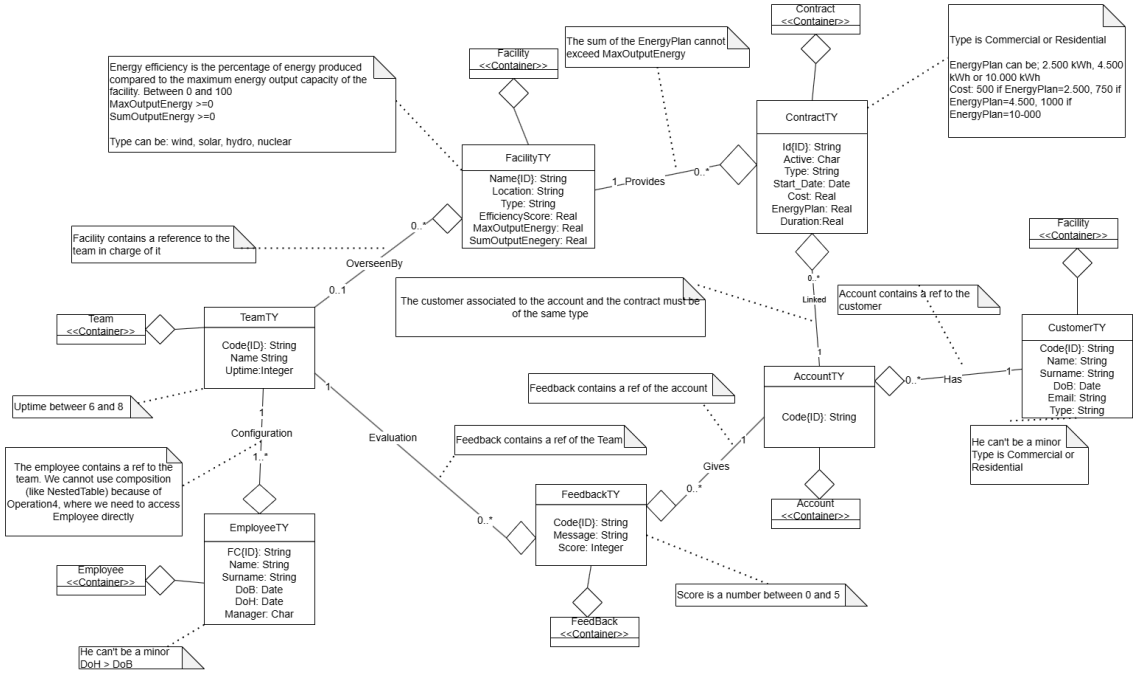


Figure 3: UML Schema

3 Implementation

3.1 Types Definition

```
CREATE OR REPLACE TYPE TeamTY AS OBJECT (
    Code VARCHAR2(20),
    Name VARCHAR2(50),
    Uptime INTEGER
)
```

```
CREATE OR REPLACE TYPE FacilityTY AS OBJECT (
    Name VARCHAR2(50),
    Location VARCHAR2(100),
    Type VARCHAR2(20),
    EfficiencyScore NUMBER,
    MaxOutputEnergy NUMBER,
    SumOutputEnergy NUMBER,
    Team REF TeamTY)
```

```
CREATE OR REPLACE TYPE EmployeeTY AS OBJECT (
    FC VARCHAR2(20),
    Name VARCHAR2(50),
    Surname VARCHAR2(50),
    DoB DATE,
    DoH DATE,
    Manager CHAR(1),
    Team REF TeamTY
)
```

```
CREATE OR REPLACE TYPE CustomerTY AS OBJECT (
    Code VARCHAR2(20),
    Name VARCHAR2(50),
    Surname VARCHAR2(50),
    Email VARCHAR2(100),
    Type VARCHAR2(20),
    DoB DATE
)
```

```
CREATE OR REPLACE TYPE AccountTY AS OBJECT (
    Code VARCHAR2(20),
    Customer REF CustomerTY
```

```
    )
```

```
CREATE OR REPLACE TYPE ContractTY AS OBJECT (
        ID VARCHAR2(20),
        Active CHAR(1),
        Type VARCHAR2(20),
        Start_Date DATE,
        Cost NUMBER,
        EnergyPlan NUMBER,
        Duration NUMBER,
        Account REF AccountTY,
        Facility REF FacilityTY
    )
```

```
    CREATE OR REPLACE TYPE FeedbackTY AS OBJECT (
        Code VARCHAR2(20),
        Message VARCHAR2(200),
        Score INTEGER,
        Account REF AccountTY ,
        Team REF TeamTY
    )
```

3.2 Tables Definition

```
    CREATE TABLE Team OF TeamTY (
        Code PRIMARY KEY,
        CONSTRAINT chk_uptime CHECK (Uptime BETWEEN 6 AND 8)
            ,
        Name NOT NULL,
        Uptime NOT NULL
    )
```

We can see that simple constraints are handled with **check**

```
    CREATE TABLE Facility OF FacilityTY (
        Name PRIMARY KEY,
        CONSTRAINT chk_facility_type CHECK (Type IN (''wind'
            ', ''solar'', ''hydro'', ''nuclear'')),
        CONSTRAINT chk_max_output_energy CHECK (
            MaxOutputEnergy >= 0),
```

```
        CONSTRAINT chk_sum_output_energy CHECK (
            SumOutputEnergy >= 0),
        Location NOT NULL,
        Type NOT NULL,
        MaxOutputEnergy NOT NULL,
        EfficiencyScore NOT NULL,
        Team REFERENCES Team ON DELETE SET NULL
    )
```

```
    CREATE TABLE Employee OF EmployeeTY (
        FC PRIMARY KEY,
        CONSTRAINT chk_manager CHECK (Manager IN (''Y'', ''N
            '')),
        Name NOT NULL,
        Surname NOT NULL,
        DoB NOT NULL,
        DoH NOT NULL,
        Manager NOT NULL,
        CONSTRAINT chk_doh_after_dob CHECK (DoH > DoB),
        Team REFERENCES Team ON DELETE SET NULL
    )
```

```
    CREATE TABLE Customer OF CustomerTY (
        Code PRIMARY KEY,
        CONSTRAINT chk_customer_type CHECK (Type IN (''
            Commercial'', ''Residential'')),
        Name NOT NULL,
        Surname NOT NULL,
        Email NOT NULL,
        Type NOT NULL,
        DoB NOT NULL
    )
```

```
    CREATE TABLE Account OF AccountTY (
        Code PRIMARY KEY,
        Customer NOT NULL REFERENCES Customer ON DELETE
            CASCADE
    )
```

```
    CREATE TABLE Contract OF ContractTY (
        ID PRIMARY KEY,
```

```
    CONSTRAINT chk_contract_type CHECK (Type IN (''
        Commercial'', ''Residential'')),
    CONSTRAINT chk_energy_plan CHECK (EnergyPlan IN
        (2500, 4500, 10000)),
    CONSTRAINT chk_cost CHECK (Cost IN (2500,4500,10000)
        ),
    CONSTRAINT chk_active CHECK (Active IN (''Y'', ''N''
        )),
    CONSTRAINT chk_cost CHECK (Cost >= 0),
    CONSTRAINT chk_duration CHECK (Duration >= 1),
    Start_Date NOT NULL,
    Cost NOT NULL,
    EnergyPlan NOT NULL,
    Duration NOT NULL,
    Active NOT NULL,
    Account NOT NULL REFERENCES Account ON DELETE
        CASCADE,
    Facility NOT NULL REFERENCES Facility ON DELETE
        CASCADE,
    Type NOT NULL
)
```

```
CREATE TABLE Feedback OF FeedbackTY (
    Code PRIMARY KEY,
    CONSTRAINT chk_feedback_score CHECK (Score BETWEEN 1
        AND 5),
    Message NOT NULL,
    Score NOT NULL,
    Account REFERENCES Account ON DELETE SET NULL,
    Team NOT NULL  REFERENCES Team ON DELETE CASCADE
)
```

3.3 Population procedure

```
CREATE OR REPLACE PROCEDURE PopulateDatabase(
    p_num_customers IN NUMBER,
    p_num_accounts IN NUMBER,
    p_num_contracts IN NUMBER,
    p_num_feedbacks IN NUMBER
) IS
```

```
BEGIN

    FOR i IN 1..50 LOOP
        INSERT INTO Team VALUES (
            TeamTY(
                'Team' || TO_CHAR(i),
                'TeamName' || TO_CHAR(i),
                ROUND(DBMS_RANDOM.VALUE(6, 8))
            )
        );
    END LOOP;



    FOR team_id IN 1..50 LOOP
        FOR i IN 1..5 LOOP
            DECLARE
                v_employee_code VARCHAR2(20);
                v_team_code     VARCHAR2(20) := 'Team'
                    || TO_CHAR(team_id);
                v_dob           DATE;
                v_doh           DATE;
                v_manager       CHAR(1);
            BEGIN

                IF i = 1 THEN
                    v_manager := 'Y';
                ELSE
                    v_manager := 'N';
                END IF;

                v_employee_code := 'FC' || TO_CHAR((
                    team_id - 1) * 5 + i);


                v_dob := ADD_MONTHS(SYSDATE, -ROUND(
                    DBMS_RANDOM.VALUE(18*12, 60*12)));
```

```
            v_doh := v_dob + ROUND(DBMS_RANDOM.VALUE
                (18*365, 60*365));

            INSERT INTO Employee VALUES (
                EmployeeTY(
                    v_employee_code,
                    'Name' || v_employee_code,
                    'Surname' || v_employee_code,
                    v_dob,
                    v_doh,
                    v_manager,
                    (SELECT REF(t) FROM Team t WHERE
                        t.Code = v_team_code)
                )
            );
        END;
    END LOOP;
END LOOP;


FOR i IN 1..100 LOOP
    INSERT INTO Facility VALUES (
        FacilityTY(
            'Facility' || TO_CHAR(i),
            'Location' || TO_CHAR(i),
            CASE MOD(i, 4)
                WHEN 0 THEN 'wind'
                WHEN 1 THEN 'solar'
                WHEN 2 THEN 'hydro'
                WHEN 3 THEN 'nuclear'
            END,
            100,
            900000000,
            0,
            (SELECT REF(t) FROM Team t WHERE t.Code
                = 'Team' || TO_CHAR(CEIL(i/2)))
        )
    );
END LOOP;
```

```
FOR i IN 1..p_num_customers LOOP
    DECLARE
        v_dob DATE;
    BEGIN

        v_dob := ADD_MONTHS(SYSDATE, -ROUND(
            DBMS_RANDOM.VALUE(18*12, 60*12)));

        INSERT INTO Customer VALUES (
            CustomerTY(
                'Customer' || TO_CHAR(i),
                'Name' || TO_CHAR(i),
                'Surname' || TO_CHAR(i),
                'email' || TO_CHAR(i) || '@example.
                    com',
                CASE WHEN MOD(i, 2) = 0 THEN '
                    Commercial' ELSE 'Residential'
                    END,
                v_dob
            )
        );
    END;
END LOOP;

FOR i IN 1..p_num_accounts LOOP
    INSERT INTO Account VALUES (
        AccountTY(
            'Account' || TO_CHAR(i),
            (SELECT REF(c) FROM Customer c WHERE c.
                Code = 'Customer' || TO_CHAR(ROUND(
                DBMS_RANDOM.VALUE(1, p_num_customers)
                )))
        )
    );
END LOOP;
```

```
FOR i IN 1..p_num_contracts LOOP
    DECLARE
        v_account_code VARCHAR2(20);
        v_customer_type VARCHAR2(20);
        v_start_date DATE;
        v_duration NUMBER;
        v_end_date DATE;
        v_active CHAR(1);
        v_energy_plan NUMBER;
        v_cost NUMBER;
    BEGIN
        v_account_code := 'Account' || TO_CHAR(ROUND
            (DBMS_RANDOM.VALUE(1, p_num_accounts)));

        SELECT c.Type
        INTO v_customer_type
        FROM Account a
        JOIN Customer c ON DEREF(a.Customer).Code =
            c.Code
        WHERE a.Code = v_account_code;
        v_start_date := SYSDATE;
        v_duration := ROUND(DBMS_RANDOM.VALUE(1, 12)
            );
        v_end_date := ADD_MONTHS(v_start_date,
            v_duration);
        IF v_end_date >= SYSDATE THEN
            v_active := 'Y';
        ELSE
            v_active := 'N';
        END IF;

        v_energy_plan := CASE MOD(i, 3)
            WHEN 0 THEN 2500
            WHEN 1 THEN 4500
            WHEN 2 THEN 10000
        END;
        IF v_energy_plan = 2500 THEN
            v_cost := 500;
        ELSIF v_energy_plan = 4500 THEN
            v_cost := 750;
```

```
            ELSE
                v_cost := 1000;
            END IF;


            INSERT INTO Contract VALUES (
                ContractTY(
                    'Contract' || TO_CHAR(i),
                    v_active,
                    v_customer_type,
                    v_start_date,
                    v_cost,
                    v_energy_plan,
                    v_duration,
                    (SELECT REF(a) FROM Account a WHERE
                        a.Code = v_account_code),
                    (SELECT REF(f) FROM Facility f WHERE
                        f.Name = 'Facility' || TO_CHAR(
                        ROUND(DBMS_RANDOM.VALUE(1, 100)))
                        )
                )
            );
        END;
    END LOOP;

    FOR i IN 1..p_num_feedbacks LOOP
        INSERT INTO Feedback VALUES (
            FeedbackTY(
                'Feedback' || TO_CHAR(i),
                'Message' || TO_CHAR(i),
                ROUND(DBMS_RANDOM.VALUE(1, 5)),
                (SELECT REF(a) FROM Account a WHERE a.
                    Code = 'Account' || TO_CHAR(ROUND(
                    DBMS_RANDOM.VALUE(1, p_num_accounts))
                    )),
                (SELECT REF(t) FROM Team t WHERE t.Code
                    = 'Team' || TO_CHAR(ROUND(DBMS_RANDOM
                    .VALUE(1, 50))))
            )
        );
```

```
    END LOOP;
END;
```

3.4 Trigger

```
CREATE OR REPLACE TRIGGER trg_employee_manager
FOR INSERT OR UPDATE ON Employee
COMPOUND TRIGGER
    v_team REF TeamTY;
    v_count NUMBER;
    status CHAR(1);
BEFORE EACH ROW IS
BEGIN
    v_team := :new.Team;
    status:=:new.Manager;
END BEFORE EACH ROW;
AFTER STATEMENT IS
BEGIN
    IF status = 'Y' THEN
        SELECT COUNT(*) INTO v_count
        FROM Employee
        WHERE (DEREF(Team)).Code = (DEREF(v_team)).Code
        AND Manager = 'Y';

        IF v_count > 1 THEN
            RAISE_APPLICATION_ERROR(-20001, 'Errore:
                Esiste già un manager per il team');
        ELSE
            DBMS_OUTPUT.PUT_LINE('Operazione eseguita
                con successo');
        END IF;
    END IF;
END AFTER STATEMENT;
END trg_employee_manager;
```

This trigger is used to check if there is already a manager in the team. If the new employee is a manager, the trigger checks if there is already a manager in the team. If there is, it raises an error, otherwise it allows the operation.

```
CREATE OR REPLACE TRIGGER trg_contract_type
BEFORE INSERT OR UPDATE ON Contract
```

```
    FOR EACH ROW
DECLARE
     v_customer_type VARCHAR2(20);
BEGIN
     SELECT DEREF(a.Customer).Type
          INTO v_customer_type
          FROM Account a
          WHERE a.Code = (DEREF(:NEW.Account)).Code;

     IF :NEW.Type = v_customer_type THEN
          DBMS_OUTPUT.PUT_LINE('Operation Completed');
     ELSE
          RAISE_APPLICATION_ERROR(-20003, 'Error: Contract
              type and customer type mismatch');
     END IF;
END;
```

This trigger is used to check if the type of the contract is the same as the type of the customer. If they are different, it raises an error, otherwise it allows the operation.

```
    CREATE OR REPLACE TRIGGER trg_deactivate_old_contracts
    FOR INSERT ON Contract
    FOLLOWS trg_contract_type
    COMPOUND TRIGGER
        v_account REF AccountTY;
        v_facility REF FacilityTY;
        v_contract_id VARCHAR2(20);
        v_status CHAR(1);
        BEFORE EACH ROW IS
        BEGIN
            v_account := :NEW.Account;
            v_facility := :NEW.Facility;
            v_contract_id := :NEW.ID;
            IF :NEW.Active = 'N' THEN
                -- Check if contract's end date is in the
                    past
                IF ADD_MONTHS(:NEW.Start_Date, :NEW.Duration
                    ) < SYSDATE THEN
                    DBMS_OUTPUT.PUT_LINE('You are inserting
                        an expired contract');
                ELSE
```

```
                DBMS_OUTPUT.PUT_LINE('Error: Contract is
                    not active');
                :NEW.Active := 'Y';
            END IF;
        END IF;
        v_status := :NEW.Active;
    END BEFORE EACH ROW;
    AFTER STATEMENT IS
    BEGIN
        IF v_status='Y' THEN
            FOR rec IN (
                SELECT ID
                FROM Contract
                WHERE Active = 'Y'
                AND Account = v_account
                AND Facility = v_facility
                AND ID <> v_contract_id
            ) LOOP
                UPDATE Contract
                SET Active = 'N'
                WHERE ID = rec.ID;

                DBMS_OUTPUT.PUT_LINE('Contract
                    deactivated');
            END LOOP;

            DBMS_OUTPUT.PUT_LINE('Trigger
                trg_deactivate_old_contracts executed');
        END IF;
    END AFTER STATEMENT;

END trg_deactivate_old_contracts;
/
```

This trigger is used to deactivate old contracts when a new contract is inserted. It checks if the new contract is active and if the end date is in the past. If the end date is in the past, the new contract is inserted as inactive. Then, it deactivates all the other contracts of the same account and facility.

```
    CREATE OR REPLACE TRIGGER trg_energy_plan
BEFORE INSERT OR DELETE OR UPDATE OF Active ON Contract
FOR EACH ROW
```

```
DECLARE
    v_facility_name Facility.Name%TYPE;
    v_current_sum NUMBER;
    v_max_output NUMBER;
BEGIN
    IF INSERTING OR UPDATING THEN
        SELECT f.Name INTO v_facility_name FROM Facility f
           WHERE REF(f) = :NEW.Facility;
    ELSE
        SELECT f.Name INTO v_facility_name FROM Facility f
           WHERE REF(f) = :OLD.Facility;
    END IF;
    SELECT SumOutputEnergy, MaxOutputEnergy
    INTO v_current_sum, v_max_output
    FROM Facility
    WHERE Name = v_facility_name
    FOR UPDATE;
    IF INSERTING THEN
        IF v_current_sum + :NEW.EnergyPlan > v_max_output
           THEN
            RAISE_APPLICATION_ERROR(-20001, 'SumOutputEnergy
                supera MaxOutputEnergy per la Facility ' ||
               v_facility_name);
        ELSE
            UPDATE Facility
            SET SumOutputEnergy = v_current_sum + :NEW.
               EnergyPlan
            WHERE Name = v_facility_name;
        END IF;
    ELSIF DELETING THEN
        UPDATE Facility
        SET SumOutputEnergy = v_current_sum - :OLD.
           EnergyPlan
        WHERE Name = v_facility_name;
    ELSIF UPDATING THEN
        IF :OLD.Active = 'Y' AND :NEW.Active = 'N' THEN
            UPDATE Facility
            SET SumOutputEnergy = v_current_sum - :OLD.
               EnergyPlan
            WHERE Name = v_facility_name;
```

```
        END IF;
    END IF;
END trg_energy_plan;
```

This trigger is used to update the sum of the output energy of the facility when a contract is inserted, deleted or updated. It checks if the sum of the output energy exceeds the maximum output energy of the facility. If it does, it raises an error, otherwise it allows the operation.

```
    CREATE OR REPLACE TRIGGER trg_efficiency_score
BEFORE UPDATE ON Facility
FOR EACH ROW
BEGIN
    IF :NEW.SumOutputEnergy = 0 THEN
        :NEW.EfficiencyScore := 100;
    ELSE
        :NEW.EfficiencyScore := 100 * :NEW.SumOutputEnergy /
            :NEW.MaxOutputEnergy;
    END IF;
END trg_efficiency_score;
```

This trigger is used to update the efficiency score of the facility when the sum of the output energy is updated. It calculates the efficiency score as the ratio between the sum of the output energy and the maximum output energy.

```
    CREATE OR REPLACE TRIGGER trg_customer_age
AFTER INSERT OR UPDATE ON Customer
FOR EACH ROW
BEGIN
    IF MONTHS_BETWEEN(SYSDATE, :NEW.DoB) < 18*12 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Error: Customer is
            a minor');
    END IF;
END;
```

```
CREATE OR REPLACE TRIGGER trg_employee_age
AFTER INSERT OR UPDATE ON Employee
FOR EACH ROW
BEGIN
    IF MONTHS_BETWEEN(SYSDATE, :NEW.DoB) < 18*12 THEN
        RAISE_APPLICATION_ERROR(-20002, 'Error: Employee is
            a minor');
    END IF;
```

```
END;
```

These two triggers are used to check if the age of the customer or the employee is less than 18 years old. If it is, it raises an error, otherwise it allows the operation.

```
    CREATE OR REPLACE TRIGGER trg_contract_cost
BEFORE INSERT OR UPDATE ON Contract
FOR EACH ROW
BEGIN
    IF UPDATING THEN
        IF (:NEW.EnergyPlan <> :OLD.EnergyPlan) OR (:NEW.
            Cost <> :OLD.Cost) THEN
            RAISE_APPLICATION_ERROR(-20003, 'Error: Energy
                plan and cost cannot be changed');
        END IF;
    END IF;

    IF INSERTING THEN
        IF :NEW.EnergyPlan = 2500 THEN
            :NEW.Cost := 500;
        ELSIF :NEW.EnergyPlan = 4500 THEN
            :NEW.Cost := 750;
        ELSE
            :NEW.Cost := 1000;
        END IF;
    END IF;
END;
```

This trigger is used to check if the energy plan and the cost of the contract are changed. If they are, it raises an error, otherwise it allows the operation. It also sets the cost of the contract based on the energy plan when a new contract is inserted.

3.5 Operations

```
--
-------------------------------------------------------------------------

-- Procedure 1: Add a new customer to the database
--
-------------------------------------------------------------------------
```

```
CREATE OR REPLACE PROCEDURE proc_register_customer (
    p_code        IN VARCHAR2,
    p_name        IN VARCHAR2,
    p_surname     IN VARCHAR2,
    p_email       IN VARCHAR2,
    p_type        IN VARCHAR2,  -- 'Commercial' o '
        Residential'
    p_dob         IN DATE
) AS
BEGIN
    INSERT INTO Customer
    VALUES (
        CustomerTY(p_code, p_name, p_surname, p_email,
            p_type, p_dob)
    );
    COMMIT;
END;
/

--
    ------------------------------------------------------------

-- Procedure 2: Add a new contract to the database
--
    ------------------------------------------------------------

CREATE OR REPLACE PROCEDURE proc_add_contract (
    p_contract_id     IN VARCHAR2,
    p_contract_type   IN VARCHAR2,  -- 'Commercial' o '
        Residential'
    p_start_date      IN DATE,
    p_energy_plan     IN NUMBER,    -- 2500, 4500 o 10000
    p_duration        IN NUMBER,
    p_account_code    IN VARCHAR2,
    p_facility_name   IN VARCHAR2
) AS
    p_cost NUMBER;
BEGIN
    IF p_energy_plan = 2500 THEN
        p_cost := 500;
```

```
        ELSIF p_energy_plan = 4500 THEN
            p_cost := 750;
        ELSE
            p_cost := 1000;
        END IF;
        INSERT INTO Contract
        VALUES (
            ContractTY(
                p_contract_id,
                'N',
                p_contract_type,
                p_start_date,
                p_cost,
                p_energy_plan,
                p_duration,
                (SELECT REF(a) FROM Account a WHERE a.Code =
                    p_account_code),
                (SELECT REF(f) FROM Facility f WHERE f.Name
                   = p_facility_name)
            )
        );
        COMMIT;
    EXCEPTION
        WHEN OTHERS THEN
            RAISE_APPLICATION_ERROR(-20011, 'Errore in
               proc_add_contract: ' || SQLERRM);
    END;
/


--
    ------------------------------------------------------------------------

-- Procedure 3: Assign a facility to a team
--
    ------------------------------------------------------------------------

CREATE OR REPLACE PROCEDURE proc_assign_facility (
    p_facility_name IN VARCHAR2,
    p_team_code     IN VARCHAR2
```

```
) AS
    v_count_team NUMBER;
    v_count_facility NUMBER;
BEGIN
    -- Check if the facility exists
    SELECT COUNT(*) INTO v_count_facility
    FROM Facility
    WHERE Name = p_facility_name;

    IF v_count_facility = 0 THEN
        RAISE_APPLICATION_ERROR(-20013, 'Error: Facility
            not found!');
    END IF;

    -- Check if the team exists
    SELECT COUNT(*) INTO v_count_team
    FROM Team
    WHERE Code = p_team_code;

    IF v_count_team = 0 THEN
        RAISE_APPLICATION_ERROR(-20014, 'Error: Team not
            found!');
    END IF;
    UPDATE Facility
    SET Team = (SELECT REF(t) FROM Team t WHERE t.Code =
        p_team_code)
    WHERE Name = p_facility_name;

    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        RAISE_APPLICATION_ERROR(-20012, 'Errore in
            proc_assign_facility: ' || SQLERRM);
END;
/
--
    ------------------------------------------------------------------

-- Function 4: Return the total energy output of a
   facility
```

```
--
    ------------------------------------------------------------------------

CREATE OR REPLACE FUNCTION func_get_facility_energy (
    p_facility_name IN VARCHAR2
) RETURN NUMBER AS
    v_total_energy NUMBER;
BEGIN

    -- Return the total energy output of the facility
    SELECT f.SumOutputEnergy
      INTO v_total_energy
      FROM Facility f
     WHERE f.Name = p_facility_name;
    RETURN v_total_energy;
EXCEPTION
    WHEN OTHERS THEN
        RAISE_APPLICATION_ERROR(-20013, 'Errore in
            func_get_facility_energy: ');
END;
/

--
    ------------------------------------------------------------------------

-- Procedure 5: Get the ranked facilities by efficiency
   score
--
    ------------------------------------------------------------------------

CREATE OR REPLACE PROCEDURE proc_get_ranked_facilities (
    p_cursor OUT SYS_REFCURSOR
) AS
BEGIN
    OPEN p_cursor FOR
      SELECT Name, EfficiencyScore
        FROM Facility
       ORDER BY EfficiencyScore DESC;
END;
/
```

4 Physical Design

Here we will discuss the physical design of the system, so the creation of indexes

4.1 Indexes for operations

4.1.1 Operation1

Explain Plan

Figure 4: Operation1

Auto-Trace

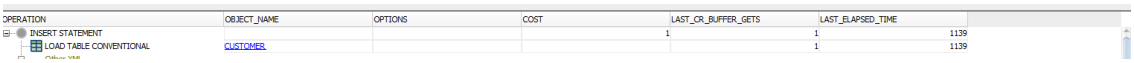


Figure 5: Operation1

Since the operation is a simple insert, it doesn't require any indexes to be created. The data will be inserted into the table as it is.

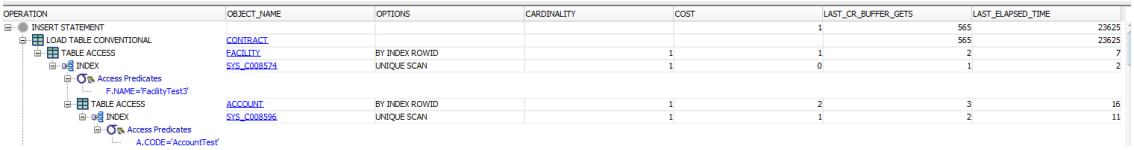
4.1.2 Operation2

Explain Plan

```
Descritto.


PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------------------------------------
Plan hash value: 84446279

-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | INSERT STATEMENT             |              |     1 |   100 |     1   (0)| 00:00:01 |
|   1 |  LOAD TABLE CONVENTIONAL     | CONTRACT     |       |       |            |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| FACILITY     |     1 |    49 |     1   (0)| 00:00:01 |
|*  3 |    INDEX UNIQUE SCAN         | SYS_C008873 |     1 |       |     1   (0)| 00:00:01 |
|   4 |   TABLE ACCESS BY INDEX ROWID| ACCOUNT      |     1 |    34 |     2   (0)| 00:00:01 |
|*  5 |    INDEX UNIQUE SCAN         | SYS_C008895 |     1 |       |     1   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("F"."NAME"='FacilityTest3')
   5 - access("A"."CODE"='AccountTest')

18 righe selezionate.
```

Figure 6: Operation2

Auto-Trace



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME
INSERT STATEMENT				1	565	23625
LOAD TABLE CONVENTIONAL	CONTRACT				565	23625
TABLE ACCESS	FACILITY	BY INDEX ROWID	1	1	2	7
INDEX	SYS_C008574	UNIQUE SCAN	1	0	1	2
Access Predicates						
F.NAME='FacilityTest3'						
TABLE ACCESS	ACCOUNT	BY INDEX ROWID	1	2	3	16
INDEX	SYS_C008596	UNIQUE SCAN	1	1	2	11
Access Predicates						
A.CODE='AccountTest'						

Figure 7: Operation2

Since the operation is a simple insert, it doesn't require any indexes to be created. The data will be inserted into the table as it is.

4.1.3 Operation3

The operation is an update, so the creation of an index is not required. However, we check for the existence of the Facility and of the Team but they are primary keys, so they are already indexed.

Explain Plan

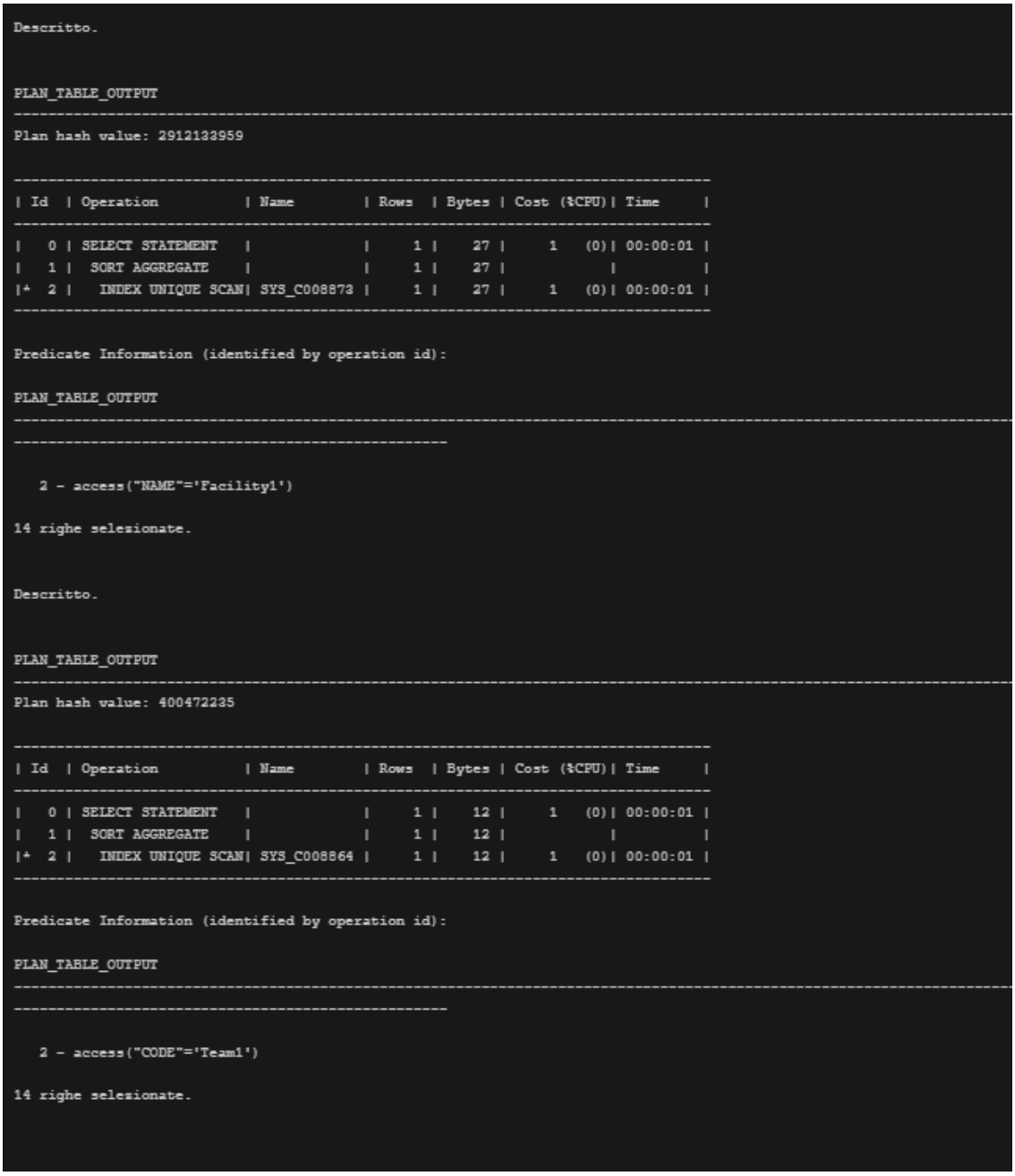
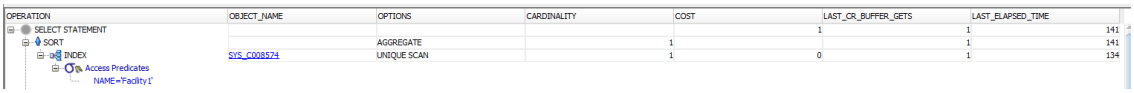


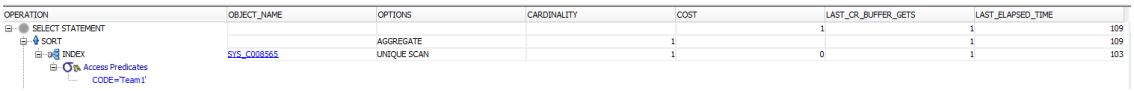
Figure 8: Operation3

Auto-Trace



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME
SELECT STATEMENT				1	1	141
SORT		AGGREGATE	1		1	141
INDEX	SYS_C008574	UNIQUE SCAN	1	0	1	134
Access Predicates						
NAME='Facility1'						

Figure 9: Facility Name Index



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME
SELECT STATEMENT				1	1	109
SORT		AGGREGATE	1		1	109
INDEX	SYS_C008565	UNIQUE SCAN	1	0	1	103
Access Predicates						
CODE='Team1'						

Figure 10: Team Code Index

4.1.4 Operation4

The first part is the selection of the oldest employee who is a manager. We can create an index on the Date of Birth (DoB) attribute of the Employee table, so that they will be ordered by DoB and we can fetch the first row.

Explain Plan

```
Descritto.


PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------------------
Plan hash value: 3201819844

-------------------------------------------------------------------------------
| Id  | Operation                | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |          |     1 |    34 |     6  (17)| 00:00:01 |
|*  1 |  VIEW                    |          |     1 |    34 |     6  (17)| 00:00:01 |
|*  2 |   WINDOW SORT PUSHED RANK|          |    50 |  4300 |     6  (17)| 00:00:01 |
|*  3 |    HASH JOIN OUTER       |          |    50 |  4300 |     5   (0)| 00:00:01 |
|*  4 |     TABLE ACCESS FULL    | EMPLOYEE |    50 |  1100 |     3   (0)| 00:00:01 |
|   5 |     TABLE ACCESS FULL    | TEAM     |    52 |  3328 |     2   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("from$_subquery$_002"."rowlimit_$$_rownumber"<=1)
   2 - filter(ROW_NUMBER() OVER ( ORDER BY "E"."DOB")<=1)
   3 - access("P000004$"."SYS_NC_OID$"(+)="E"."SYS_NC00010$")
   4 - filter("E"."MANAGER"='Y')

Note

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------------------
-----
   - dynamic statistics used: dynamic sampling (level=2)

24 righe selezionate.
```

Figure 11: Operation4

Auto-Trace

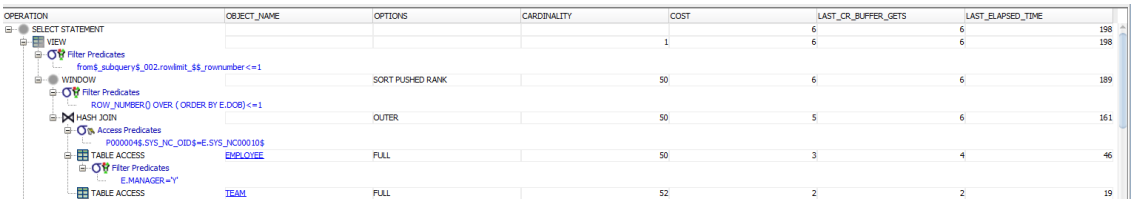
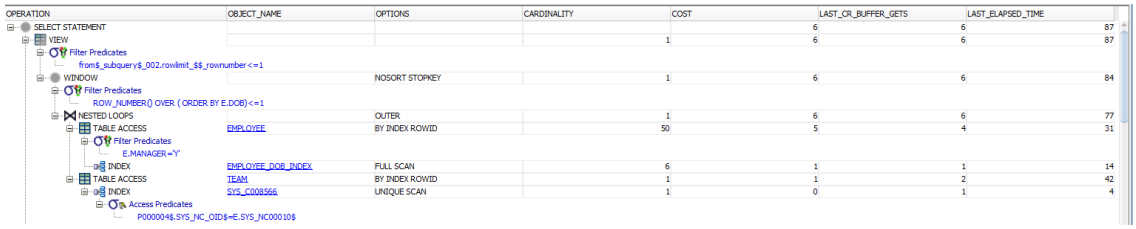


Figure 12: Autotrace without Index

By using the autotrace we can see that here we are not using any index. Let's create the index

```
CREATE INDEX Employee_DoB_Index ON Employee(DoB);
```



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME
SELECT STATEMENT				6	6	87
VIEW			1	6	6	87
Filter Predicates						
from$_subquery$_002.rowlimit_$$_rownumber<=1						
WINDOW		NOSORT STOPKEY	1	6	6	84
Filter Predicates						
ROW_NUMBER() OVER (ORDER BY E.DOB)<=1						
NESTED LOOPS		OUTER	1	6	6	77
TABLE ACCESS	EMPLOYEE	BY INDEX ROWID	50	5	4	31
Filter Predicates						
E.MANAGER='Y'						
INDEX	EMPLOYEE_DOB_INDEX	FULL SCAN	6	1	1	14
TABLE ACCESS	TEAM	BY INDEX ROWID	1	1	2	42
INDEX	SYS_C008566	UNIQUE SCAN	1	0	1	4
Access Predicates						
P000004$.SYS_NC_OID$=E.SYS_NC00010$						

Figure 13: Autotrace with Index

We can see that now we are using the Employee_DoB_Index index to fetch the first row (the Autotrace shows UNIQUE SCAN). The second part of the operation is a simple sselection. It uses Facility.Name with is already indexed.

4.1.5 Operation5

Operation five is a selection sorted by FacilityEfficiency. However, even with an index on that attribute, the Autotrace shows that the index is not used.

Explain Plan

```
Descritto.


PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------
Plan hash value: 3030065311

-------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          |   105 |  4200 |     3  (34)| 00:00:01 |
|   1 |  SORT ORDER BY     |          |   105 |  4200 |     3  (34)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| FACILITY |   105 |  4200 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Note

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------
-----
   - dynamic statistics used: dynamic sampling (level=2)

13 righe selezionate.
```

Figure 14: Operation5

Auto-Trace

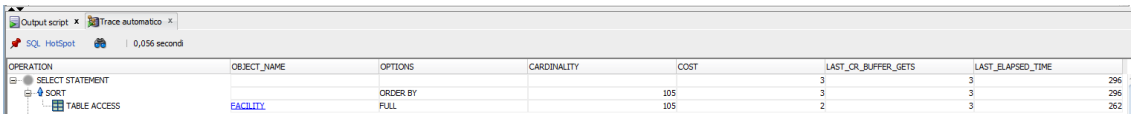


Figure 15: Autotrace

5 Web Interface

The web interface is a simple web application that allows users to interact with the system. It allows users to execute the 5 operations previously described.

5.1 Backend

It is implemented using the Flask web framework of Python.

```
    # app.py
from flask import Flask, render_template, request, redirect,
    url_for, flash
import oracledb
import datetime

app = Flask(__name__)
app.secret_key = 'supersecretkey'  # per flash messages

# Configurations parameters for the Oracle DB
DB_USER = 'System'
DB_PASSWORD = 'password123'
DB_SID = 'localhost:1521/xe'


def get_db_connection():
    try:
        connection = oracledb.connect(user=DB_USER, password
           =DB_PASSWORD, dsn=DB_SID)
        return connection
    except Exception as e:
        print("Errore nella connessione al DB:", e)
        return None

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

# Operation 1: Register a new customer
@app.route('/register_customer', methods=['GET', 'POST'])
def register_customer():
    if request.method == 'POST':
```

```
        code = request.form.get('code')
        name = request.form.get('name')
        surname = request.form.get('surname')
        email = request.form.get('email')
        cust_type = request.form.get('cust_type')
        dob_str = request.form.get('dob')
        try:
            dob = datetime.datetime.strptime(dob_str, '%Y-%m
                -%d').date()
        except Exception as e:
            flash("Birth date non valida", "danger")
            return redirect(url_for('register_customer'))

        conn = get_db_connection()
        if conn is None:
            flash("Connection error", "danger")
            return redirect(url_for('register_customer'))
        try:
            cur = conn.cursor()
            cur.callproc("proc_register_customer", [code,
                name, surname, email, cust_type, dob])
            conn.commit()
            flash("Customer added successfully", "success")
        except oracledb.DatabaseError as e:
            flash(f"Error during customer registration: {e}"
                , "danger")
        finally:
            cur.close()
            conn.close()
        return redirect(url_for('index'))
    return render_template('register_customer.html')

# Operation 2: Add a new energy contract
@app.route('/add_contract', methods=['GET', 'POST'])
def add_contract():
    if request.method == 'POST':
        contract_id = request.form.get('contract_id')
        contract_type = request.form.get('contract_type')
        start_date_str = request.form.get('start_date')
        energy_plan = float(request.form.get('energy_plan'))
```

```
        duration = float(request.form.get('duration'))
        account_code = request.form.get('account_code')
        facility_name = request.form.get('facility_name')
        try:
            start_date = datetime.datetime.strptime(
                start_date_str, '%Y-%m-%d').date()
        except Exception as e:
            flash("Not a valid date", "danger")
            return redirect(url_for('add_contract'))

        conn = get_db_connection()
        if conn is None:
            flash("Error during connection to the database",
                "danger")
            return redirect(url_for('add_contract'))
        try:
            cur = conn.cursor()

            cur.callproc("proc_add_contract", [contract_id,
                contract_type,
                                               start_date,

                                                   energy_plan
                                                   ,
                                                   duration
                                                   ,
                                               account_code
                                                   ,
                                                   facility_name
                                                   ])
            conn.commit()
            flash("Contract added successfully", "success")
        except Exception as e:
            flash(f"Error during contract addition: {e}", "
                danger")
        finally:
            cur.close()
            conn.close()
        return redirect(url_for('index'))
    return render_template('add_contract.html')
```

```
# Operation 3: Assign a facility to a management team
@app.route('/assign_facility', methods=['GET', 'POST'])
def assign_facility():
    if request.method == 'POST':
        facility_name = request.form.get('facility_name')
        team_code = request.form.get('team_code')
        conn = get_db_connection()
        if conn is None:
            flash("Error during connection to the database",
                "danger")
            return redirect(url_for('assign_facility'))
        try:
            cur = conn.cursor()
            cur.callproc("proc_assign_facility", [
                facility_name, team_code])
            conn.commit()
            flash("Facility assigned successfully", "success
                ")
        except Exception as e:
            flash(f"Error during facility assignment: {e}",
                "danger")
        finally:
            cur.close()
            conn.close()
        return redirect(url_for('index'))
    return render_template('assign_facility.html')

# Operation 4: View the total energy output of a facility
    managed by the eldest employee
@app.route('/view_facility_energy', methods=['GET', 'POST'])
def view_facility_energy():
    facilities = []
    energy_output = None

    # 1. Query to retrieve the oldest manager's team code
    conn = get_db_connection()
    if conn is None:
        flash("Error during connection to the database", "
            danger")
```

```
        return redirect(url_for('index'))
    try:
        cur = conn.cursor()
        query_oldest_manager = """
            SELECT DEREF(e.Team).Code AS team_code
              FROM Employee e
             WHERE e.Manager = 'Y'
             ORDER BY e.DoB ASC
             FETCH FIRST 1 ROWS ONLY
        """
        cur.execute(query_oldest_manager)
        result = cur.fetchone()
        if result is None:
            flash("No manager found", "danger")
        else:
            team_code = result[0]
            # 2. Query to retrieve the facilities managed by
                the oldest manager
            query_facilities = """
                SELECT f.Name
                  FROM Facility f
                 WHERE DEREF(f.Team).Code = :team_code
            """
            cur.execute(query_facilities, {'team_code':
               team_code})
            facilities = [row[0] for row in cur.fetchall()]
    except Exception as e:
        flash(f"Error during data retrieval: {e}", "danger")
    finally:
        cur.close()
        conn.close()

    # 3. Retrieve the total energy output of the selected
       facility
    if request.method == 'POST':
        selected_facility = request.form.get('facility')
        conn = get_db_connection()
        if conn is None:
            flash("Error during connection to the database",
                "danger")
```

```
            return redirect(url_for('view_facility_energy'))
        try:
            cur = conn.cursor()
            # Chiamata alla funzione SQL per ottenere il
               totale dell'energia in output
            energy_output = cur.callfunc("
               func_get_facility_energy", oracledb.NUMBER, [
               selected_facility])
            flash(f"Total Energy Output for {
               selected_facility}: {energy_output}", "
               success")
        except Exception as e:
            flash(f"Error during data retrieval: {e}", "
               danger")
        finally:
            cur.close()
            conn.close()

    return render_template("view_facility_energy.html",
       facilities=facilities, energy_output=energy_output)

# Operation 5: Print a ranked list of facilities based on
   their efficiency scores
@app.route('/ranked_facilities')
def ranked_facilities():
    facilities = []
    conn = get_db_connection()
    if conn is None:
        flash("Errore di connessione al database", "danger")
        return redirect(url_for('index'))
    try:
        cur = conn.cursor()
        out_cursor = cur.var(oracledb.CURSOR)
        cur.callproc("proc_get_ranked_facilities", [
           out_cursor])
        result_cursor = out_cursor.getvalue()
        facilities = result_cursor.fetchall()
    except Exception as e:
        flash(f"Errore durante il recupero dei dati: {e}", "
           danger")
```

```
    finally:
        cur.close()
        conn.close()
    return render_template('ranked_facilities.html',
        facilities=facilities)

if __name__ == '__main__':
    app.run(debug=True)
```

5.2 Frontend

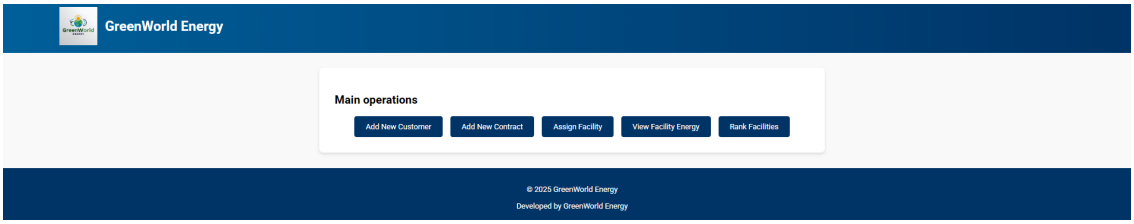


Figure 16: Homepage

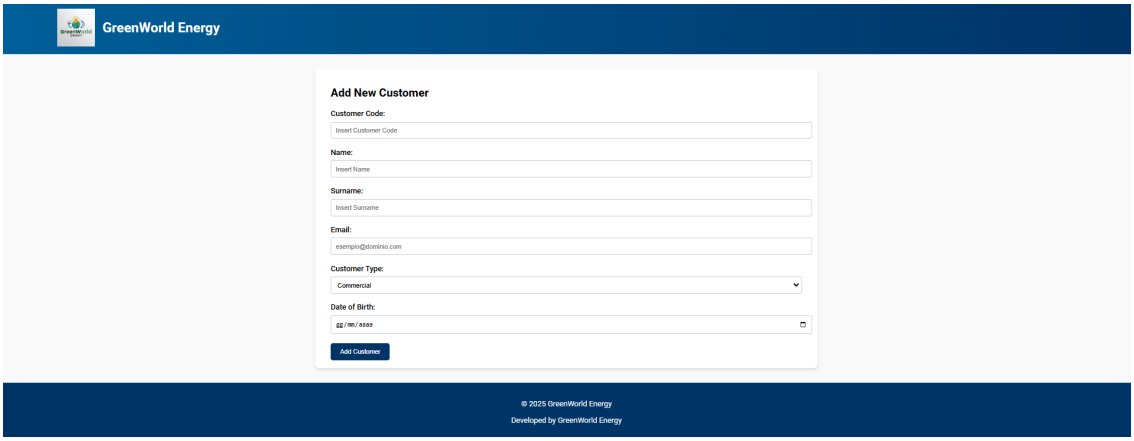


Figure 17: Register a new customer

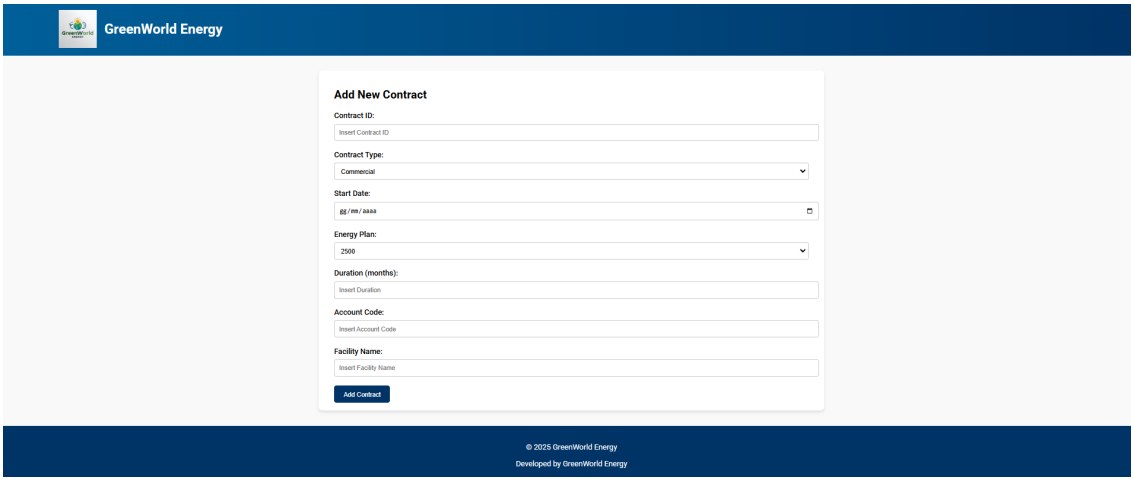


Figure 18: Add a new energy contract

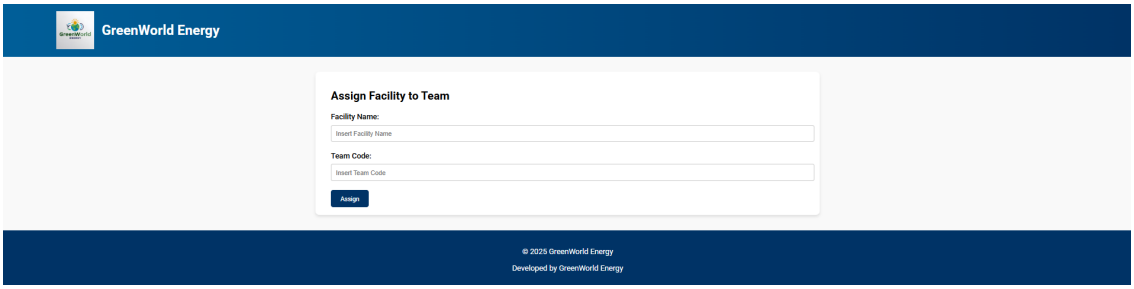


Figure 19: Assign a facility to a management team

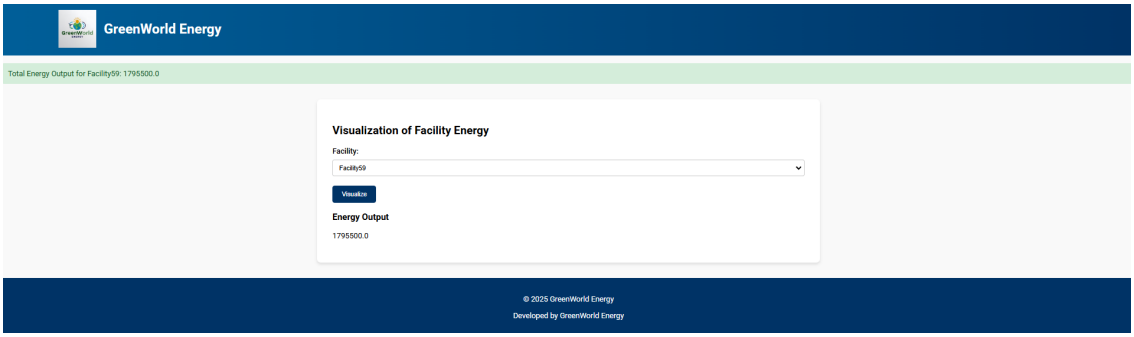


Figure 20: View the total energy output of a facility managed by the eldest employee

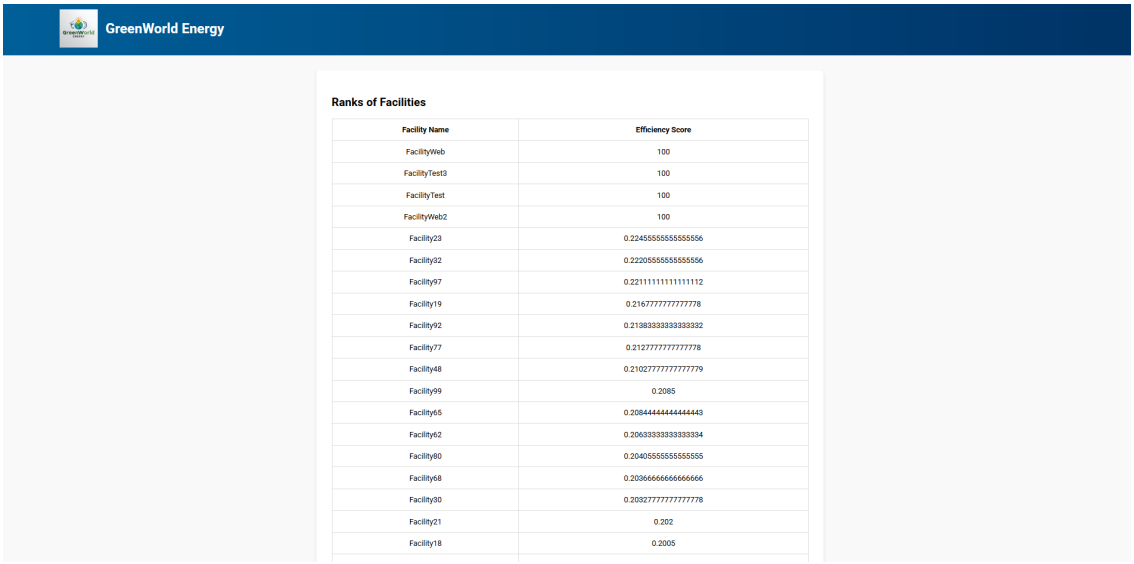


Figure 21: Print a ranked list of facilities based on their efficiency scores