UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

Computer Science Department

Master Degree in Computer Science

Practical Part of **Database Systems**

Brightway: Technical Documentation

Student: Student ID:

Mattia Curri (*m.curri8@studenti.uniba.it*) **832437**

ACADEMIC YEAR **2024-2025**

Contents

1 Requirements

	"Brightway"
1.	The company "Brightway" manages decentralized logistics operations through several operational centers
2.	distributed across regions, each responsible for handling local storage and shipments. Each operational center is
3.	characterized by a name, address, city/province, and number of employees. The company offers customized
4.	warehouse management services, including long-term storage and expedited shipping. Orders can be placed by
5.	customers via phone, email, or directly through the company's online platform.
6.	Each customer may have one or more business accounts, each identified by a unique code. Every order is associated
7.	with a single business account and includes details such as type, date, cost, and customer information. Orders can
8.	be of three types: regular, urgent, or bulk (large quantities). Operational centers have management teams, each
9.	identified by a unique code, name, and the number of operations handled. Teams consist of specialized personnel,
10.	and the number of members may vary depending on the required workload.
11	Additionally, the company maintains a performance evaluation system that assigns a score to each team based on
12	delivery times and customer feedback. Customers can be classified as individual or business, each identified by a
13	unique alphanumeric code, with contact details and order history.

2 | Conceptual Design

Requirements Analysis

Reorganize sentences for specific concepts

General Phrases

We want to create a database that manages decentralized logistics operations through several operational centers, storing information about orders, teams, and customers. The company offers customized warehouse management services, including long-term storage and expedited shipping.

Phrases related to Operational Centers

Operational centers are distributed across regions, each responsible for handling local storage and shipments.
For each operational center, we will hold name, address, city/province, and number of employees.
Operational centers have management teams.

Phrases related to Orders

Orders can be placed by customers via phone, email, or online.
For each order, we will hold type, date, cost, and customer information.
Every order is associated with a single business account.
Orders can be of three types: regular, urgent, or bulk (large quantities).

Phrases related to Business Accounts

Each business account will be identified by a unique code.
Each customer may have one or more business accounts.
Every order is associated with a single business account.

Phrases related to Teams

For each team, we will identify them via a unique code, and we will hold name and number of orders handled.
Teams consist of employees, and the number of employees may vary depending on the required workload.
The company maintains a performance evaluation system that assigns a score to each team based on delivery times and customer feedback.

Phrases related to Customers

Each customer may have one or more business accounts.
Customers can be classified as individual or business, each identified by a unique alphanumeric code, with contact details and order history.

Phrases related to Employees

Teams consist of employees, and the number of employees may vary depending on the required workload.

Level of abstraction

For **customer information**, we consider *name*, *surname*, and *date of birth* for individual customers, and *company name* and *address* for business customers, where the address consists of *street*, *civic number*, *city*, *province*, *region*, and *state*.

For **contact details**, we consider *phone number* and *email*.

Specialized personnel is replaced by *employees*.

Number of members is replaced by *number of employees*.

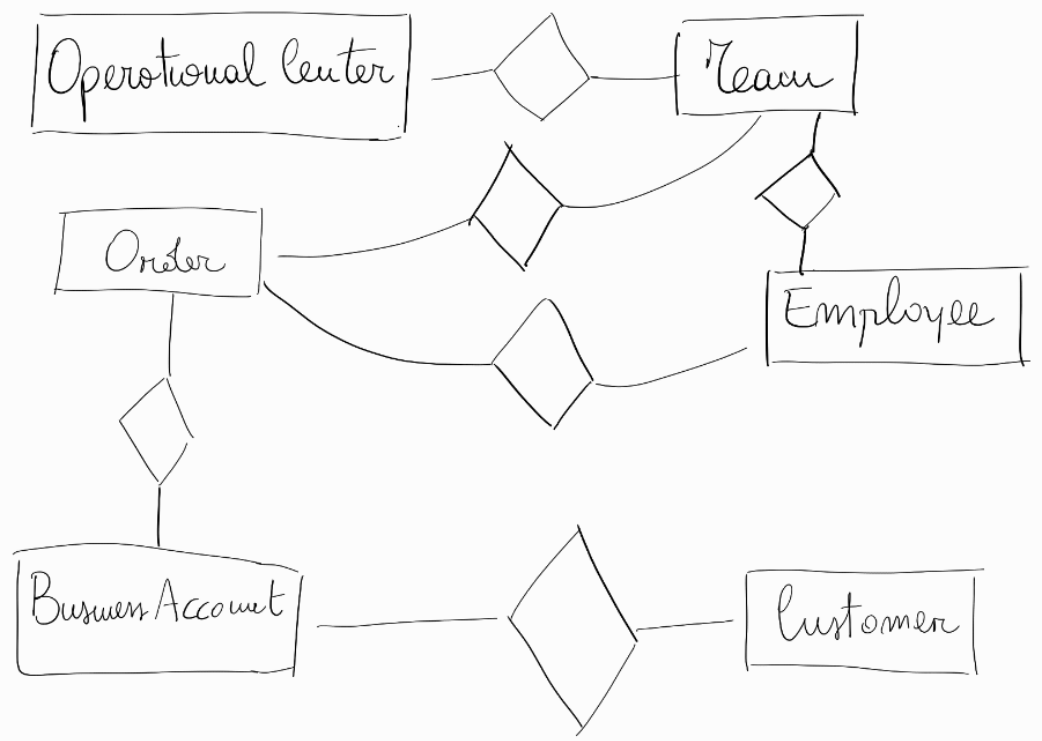
Number of operations is replaced by *number of orders*.

Glossary of terms

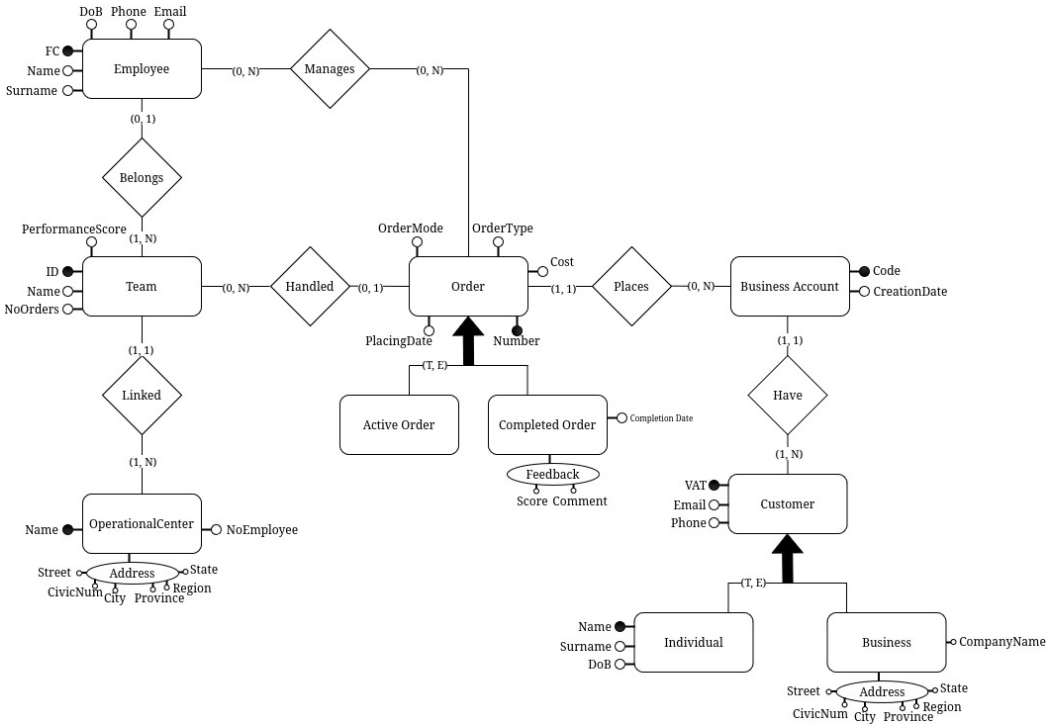
Term	Description	Synonyms	Connections
Operational Center	Decentralized locations handling local storage and shipments. Characterized by name, address, city/province, and number of employees.	/	Team
Order	Requests for services or goods from customers, identified by type (regular, urgent, or bulk). Includes type, date, cost, and customer details.	Operation	Customer, Business Account, Team
Business Account	Accounts tied to customers, containing unique codes and details like orders and customer type (individual or business).	Account	Customer, Order
Team	Groups of employees linked to an operational center, evaluated on delivery times and customer feedback.	Management Team	Operational Center, Order, Employee
Customer	Individuals or businesses placing orders. Identified by unique alphanumeric codes, contact details, and order history.	Individual, Business	Order, Business Account
Employee	Specialized personnel working in teams and handling orders.	Specialized Personnel	Team

Table 2.1: Glossary of terms

Skeleton ER Schema



Final ER Schema



- There are three redundancies: `NoOrders` and `PerformanceScore` in `Team` and `NoEmployee` in `Operational Center`.
- There are two generalizations (both total and exclusive) for `Order` and `Customer`.

Business Rules

- Employees linked to an order must be in the same team that handle the order.
- Number of employees is the sum of all employees in a team.
- Number of orders is the sum of all orders handled by a team.
- Completion date cannot be before placing date.
- Feedback of CompletedOrder must be between 1 and 5.
- PerformanceScore is computed as the mean of all feedbacks.
- A team has a maximum of 8 members.
- OrderType must be of three types: regular, urgent, or bulk.
- OrderMode must be of three types: phone, email, or online.
- A feedback cannot be given before the order is completed.
- A team must be assigned before an order is completed.
- A team cannot be changed after an order is completed.
- Employees of a completed order cannot be changed.

3 | Logical Design

Volumes Table

We will consider a span of a month to evaluate the volumes of the entities.

Concept	Type	Computation	Final Volume
Operational Center	E	15 (assuming 10 team per `Operational Center`)	15
Linked To	R	150 (same as `Team`)	150
Team	E	150 (*given*)	150
Handled By	R	45000 (same as (assigned) `Order`)	45000
Order	E	300 op/m $\cdot$ 150 + 100 not assigned	45100
Belongs To	R	150 `Team` $\cdot$ 6.4 current `Employee`	960
Employee	E	80% of (150 $\cdot$ 8) + 140 past `Employee`	1100
Manages	R	45000 $\cdot$ 3 (assuming 3 `Employee` working on an `Order`)	135000
Places	R	45100 (same as `Order`)	45100
Business Account	E	$\frac{45100\ \texttt{Order}}{1.5\ \texttt{Order/month}}$	30000
Have	R	30000 (same as `Business Account`)	30000
Customer	E	$\frac{30000\ \texttt{Business Account}}{1.5\ \texttt{Business Account/Customer}}$	20000
Active Order	E	20% of 45000 + 100 not assigned	9100
Completed Order	E	80% of 45000	36000
Individual	E	90% of `Customer`	18000
Business	E	10% of `Customer`	2000

Operations Analysis

Operation	Type	Frequency
Operation 1	I	10/day
Operation 2	I	1000/day
Operation 3	I	500/day
Operation 4	I	200/day
Operation 5	I	20/day

Table 3.1: Operations frequency

Note: we will double count the cost of writing operations.

Operation 1: Register a new customer

Concept	Type	No. Access	Access Type
Customer	E	1	W
Have	R	1	W
Business Account	E	1	W

Operation cost: 6 accesses $\cdot$ 10 = 60 accesses/day

Operation 2: Add a new order

Concept	Type	No. Access	Access Type
Order	E	1	W
Places	R	1	W

Operation cost: 4 accesses $\cdot$ 1000 = 4000 accesses/day

Operation 3: Assign an order to a management team

Access *without* redundancy `Team.NoOrders`:

Concept	Type	No. Access	Access Type
Handled By	R	1	W
Manages	R	3	W

Access *with* redundancy `Team.NoOrders`:

Concept	Type	No. Access	Access Type
Handled By	R	1	W
Team	E	1	R
Team	E	1	W
Manages	R	3	W

Operation cost (*without redundancy*): 8 accesses $\cdot$ 500 = 4000 accesses/day

Operation cost (*with redundancy*): 11 accesses $\cdot$ 500 = 5500 accesses/day

Operation 4A: View the total number of operations handled by a specific team

Access *with* redundancy `Team.NoOrders`:

Concept	Type	No. Access	Access Type
Team	E	1	R

Access *without* redundancy `Team.NoOrders`:

Concept	Type	No. Access	Access Type
Handled By	R	300	R

Operation cost (*without redundancy*): 300 accesses $\cdot$ 200 = 60000 accesses/day

Operation cost (*with redundancy*): 1 access $\cdot$ 200 = 200 accesses/day

Operation 4B: Show the total cost of the orders handled by a specific team

Access *with* redundancy `Team.NoOrders`:

Concept	Type	No. Access	Access Type
Team	E	1	R

Access *without* redundancy `Team.NoOrders`:

Concept	Type	No. Access	Access Type
Handled	R	300	R
Order	E	300	R

Operation cost (*without redundancy*): $600 \text{ accesses} \cdot 200 = 120000 \text{ accesses/day}$

Operation cost (*with redundancy*): $1 \text{ access} \cdot 200 = 200 \text{ accesses/day}$

Operation 5: Print a list of teams sorted by their performance score

Access *with* redundancy `Team.PerformanceScore`:

Concept	Type	No. Access	Access Type
Team	E	150	R

Access *without* redundancy `Team.PerformanceScore`:

Concept	Type	No. Access	Access Type
Team	E	150	R
Handled By	R	45000	R
Order	E	45000	R

Operation cost (*without redundancy*): $90150 \text{ accesses} \cdot 20 = 1803000 \text{ accesses/day}$

Operation cost (*with redundancy*): $150 \text{ accesses} \cdot 20 = 3000 \text{ accesses/day}$

Redundancy Analysis

- `OperationalCenter.NoEmployees`: Since no operations utilize this attribute, we decide to **eliminate** this redundancy.

- `Team.NoOrders`: The analysis shows 5,900 accesses with redundancy versus 184,000 accesses without redundancy when combining Operations (3), (4A), and (4B). Based on this significant difference, we decide to **maintain** this redundancy.

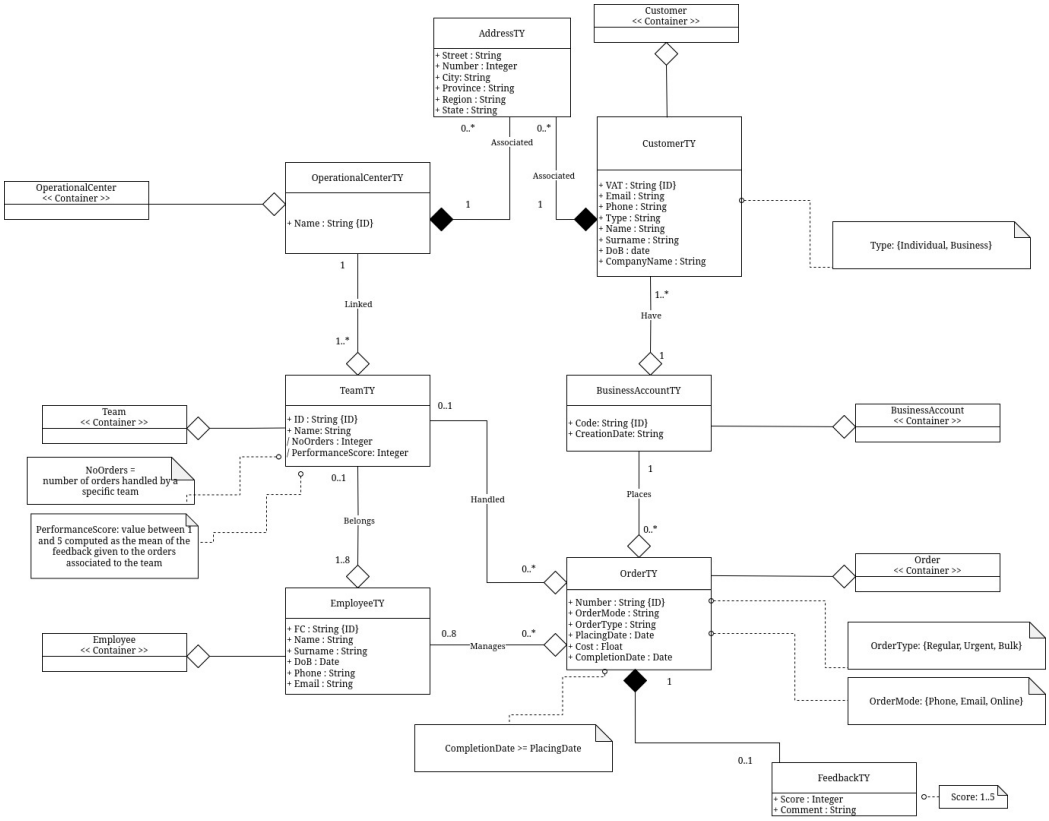
- `Team.PerformanceScore`: Operation (5) requires 3,000 accesses with redundancy compared to 1,803,000 accesses without it. Given this substantial performance impact, we decide to **maintain** this redundancy.

Partitioning and Merging

- Merging `Active Order` and `Completed Order` into `Order`: since no operations require distinguishing between these two entities, we can **merge** them into a single entity.

- Merging `Individual` and `Business` into `Customer`: since no operations require distinguishing between these two types of customers, we can **merge** them into a single entity.

Restructured Schema



Other Details

- When an Operational Center is deleted, the associated teams are also deleted, causing dereferencing of the orders linked to these teams.

- When a Customer is deleted, the associated business accounts are also deleted, causing dereferencing of the orders linked to these accounts.

- As a result of the previous observations, an order can be deleted only if there are not information for the business account (order history) or for the feedback computation.

- The existence of a completion date in a order mark the order as completed, otherwise is active.

- After a team is updated on an order, the list of employees is cleared.

- We use `AddressTY` to define the composite attribute `Address` in `OperationalCenterTY` and `CustomerTY`.

4 | Implementation

Types definition

```
CREATE OR REPLACE TYPE AddressTY AS OBJECT (
    street VARCHAR2(50),
    civicNum NUMBER,
    city VARCHAR2(50),
    province VARCHAR2(50),
    region VARCHAR2(50),
    state VARCHAR2(50)
);
```

```
CREATE OR REPLACE TYPE OperationalCenterTY AS OBJECT (
    name VARCHAR2(50),
    address AddressTY
);
```

```
CREATE OR REPLACE TYPE TeamTY AS OBJECT (
    ID VARCHAR2(32),
    name VARCHAR2(20),
    numOrder NUMBER,
    performanceScore NUMBER(4, 2),
    operationalCenter ref OperationalCenterTY
);
```

```
CREATE OR REPLACE TYPE EmployeeTY AS OBJECT (
    FC VARCHAR2(16),
    name VARCHAR2(20),
    surname VARCHAR2(20),
    dob DATE,
    phone VARCHAR2(14),
    email VARCHAR2(50),
    team ref TeamTY
);
```

```
CREATE OR REPLACE TYPE FeedbackTY AS OBJECT (
```

```
    score NUMBER(1),
    commentF VARCHAR2(1000)
);
```

```
CREATE OR REPLACE TYPE CustomerTY AS OBJECT (
    VAT VARCHAR2(11),
    phone VARCHAR2(14),
    email VARCHAR2(50),
    type VARCHAR2(10),
    name VARCHAR2(20),
    surname VARCHAR2(20),
    dob DATE,
    companyName VARCHAR2(50),
    address AddressTY
) NOT FINAL;
```

```
CREATE OR REPLACE TYPE BusinessAccountTY AS OBJECT (
    CODE VARCHAR2(32),
    creationDate DATE,
    customer ref CustomerTY
);
```

```
CREATE OR REPLACE TYPE EmployeeVA AS VARRAY(8) OF REF EmployeeTY;
```

```
CREATE OR REPLACE TYPE OrderTY AS OBJECT (
    ID VARCHAR2(32),
    placingDate DATE,
    orderMode VARCHAR2(6),
    orderType VARCHAR2(7),
    cost NUMBER(10, 2),
    businessAccount ref BusinessAccountTY,
    team ref TeamTY,
    employees EmployeeVA,
    completionDate DATE,
    feedback FeedbackTY
);
```

Table definition

```
CREATE TABLE OperationalCenterTB OF OperationalCenterTY (
    name PRIMARY KEY,
    address NOT NULL
);
```

```
CREATE TABLE TeamTB OF TeamTY (
    ID PRIMARY KEY,
    name NOT NULL,
    numOrder check (numOrder >= 0),
    performanceScore check (performanceScore between 1 and 5),
    operationalCenter NOT NULL
);
```

```
CREATE TABLE EmployeeTB OF EmployeeTY (
    FC PRIMARY KEY CHECK (LENGTH(FC) = 16),
    name NOT NULL,
    surname NOT NULL,
    dob NOT NULL,
    phone NOT NULL,
    email NOT NULL
);
```

```
CREATE TABLE CustomerTB OF CustomerTY (
    VAT PRIMARY KEY CHECK (LENGTH(VAT) = 11),
    phone NOT NULL,
    email NOT NULL,
    type NOT NULL CHECK (type IN ('individual', 'business'))
);
```

```
CREATE TABLE BusinessAccountTB OF BusinessAccountTY (
    CODE DEFAULT RAWTOHEX(SYS_GUID()) PRIMARY KEY,
    creationDate NOT NULL,
    customer NOT NULL
);
```

```
CREATE TABLE OrderTB OF OrderTY (
    ID DEFAULT RAWTOHEX(SYS_GUID()) PRIMARY KEY,
    placingDate NOT NULL,
    orderMode NOT NULL CHECK (orderMode IN ('online', 'phone', 'email'
        )),
    orderType NOT NULL CHECK (orderType IN ('regular', 'urgent', 'bulk
        ')),
    cost NOT NULL CHECK (cost > 0),
    businessAccount NOT NULL,

    check (placingDate <= completionDate),
    check (feedback.score between 1 and 5)
);
```

5 | Trigger Implementation

CheckOrderInsertOrUpdate

This trigger enforces the following logical constraints on orders:

- Feedback cannot be provided without a completion date.
- A team must be assigned before the completion date.
- The team cannot be changed after the order is completed.
- All employees in the same order must belong to the same team.
- Employees associated with a completed order cannot be updated.
- If an order has employees but no team, assign the team based on the employees' team.
- Ensures that feedback score, if feedback is provided, is not null.

```
CREATE OR REPLACE TRIGGER CheckOrderInsertOrUpdate
BEFORE INSERT OR UPDATE ON OrderTB
FOR EACH ROW
DECLARE
    cnt NUMBER;
BEGIN

    IF :NEW.completionDate IS NULL THEN
        IF :NEW.feedback IS NOT NULL THEN
            RAISE_APPLICATION_ERROR(-20001, 'Feedback cannot be given
               without completion date');
        END IF;
    END IF;


    IF INSERTING THEN
        IF :NEW.team IS NULL AND :NEW.completionDate IS NOT NULL THEN
            RAISE_APPLICATION_ERROR(-20010, 'Team must be assigned
               before completion date');
        END IF;
```

```
END IF;


IF UPDATING THEN
    IF :OLD.completionDate IS NOT NULL AND
        ((:OLD.team IS NULL AND :NEW.team IS NOT NULL) OR
        (:OLD.team IS NOT NULL AND :NEW.team IS NULL) OR
        (:OLD.team IS NOT NULL AND :NEW.team IS NOT NULL AND :NEW.
           team <> :OLD.team)) THEN
        RAISE_APPLICATION_ERROR(-20011, 'Team cannot be changed
           after order completion');
    END IF;
END IF;


IF :NEW.employees IS NOT NULL AND :NEW.employees.COUNT > 0 AND :
   NEW.team IS NOT NULL THEN
    SELECT COUNT(*) INTO cnt FROM TABLE(:NEW.employees) emp_ref
    JOIN EmployeeTB e ON (emp_ref.column_value = REF(e))
    WHERE e.team <> :NEW.team;

    IF cnt > 0 THEN
        RAISE_APPLICATION_ERROR(-20007, 'Employee of a different
           team in the same order detected');
    END IF;
END IF;


IF UPDATING THEN
    IF :OLD.completionDATE IS NOT NULL THEN
        IF :NEW.employees IS NOT NULL OR (:OLD.employees IS NOT
           NULL AND :NEW.employees IS NULL) THEN
            RAISE_APPLICATION_ERROR(-20008, 'Cannot update
               employees of a completed order');
        END IF;
    END IF;
END IF;


IF :NEW.team IS NULL AND :NEW.employees IS NOT NULL AND :NEW.
   employees.COUNT > 0 THEN
    SELECT e.team INTO :NEW.team
    FROM TABLE(:NEW.employees) emp_ref
    JOIN EmployeeTB e ON (emp_ref.column_value = REF(e))
```

```
        FETCH FIRST 1 ROW ONLY;
    END IF;


    IF :NEW.feedback IS NOT NULL AND :NEW.feedback.score IS NULL THEN
        RAISE_APPLICATION_ERROR(-20020, 'Feedback score cannot be null
            ');
    END IF;
END;
```

CheckCustomerType

Enforces that "individual" customers cannot have business data and vice versa:

```
CREATE OR REPLACE TRIGGER CheckCustomerType
BEFORE INSERT OR UPDATE ON CustomerTB
FOR EACH ROW
BEGIN
    IF :NEW.type = 'individual' THEN
        IF :NEW.companyName IS NOT NULL THEN
            RAISE_APPLICATION_ERROR(-20002, 'Individual customers
                cannot have a company name');
        END IF;
        IF :NEW.address IS NOT NULL THEN
            RAISE_APPLICATION_ERROR(-20003, 'Individual customers
                cannot have an address');
        END IF;
    ELSIF :NEW.type = 'business' THEN
        IF :NEW.name IS NOT NULL THEN
            RAISE_APPLICATION_ERROR(-20004, 'Business customers cannot
                have a name');
        END IF;
        IF :NEW.surname IS NOT NULL THEN
            RAISE_APPLICATION_ERROR(-20005, 'Business customers cannot
                have a surname');
        END IF;
        IF :NEW.dob IS NOT NULL THEN
            RAISE_APPLICATION_ERROR(-20006, 'Business customers cannot
                have a date of birth');
        END IF;
    END IF;
END;
```

UpdateNumOrdersBeforeInsert

Increments the numOrder attribute of a team before inserting a new order:

```
CREATE OR REPLACE TRIGGER UpdateNumOrdersBeforeInsert
BEFORE INSERT ON OrderTB
FOR EACH ROW
BEGIN
    IF :NEW.team IS NOT NULL THEN
        UPDATE TeamTB t
            SET t.numOrder = t.numOrder + 1
            WHERE REF(t) = :NEW.team;
    END IF;
END;
```

UpdateNumOrdersBeforeDelete

Decrements the numOrder attribute of a team before deleting an order:

```
CREATE OR REPLACE TRIGGER UpdateNumOrdersBeforeDelete
BEFORE DELETE ON OrderTB
FOR EACH ROW
BEGIN
    IF :OLD.team IS NOT NULL THEN
        UPDATE TeamTB t
            SET t.numOrder = t.numOrder - 1
            WHERE REF(t) = :OLD.team;
    END IF;
END;
```

UpdateNumOrdersBeforeUpdate

Updates the numOrder attribute of a team before updating an order:

```
CREATE OR REPLACE TRIGGER UpdateNumOrdersBeforeUpdate
BEFORE UPDATE OF team ON OrderTB
FOR EACH ROW
BEGIN
    IF :OLD.team IS NOT NULL THEN
        UPDATE TeamTB t
            SET t.numOrder = t.numOrder - 1
            WHERE REF(t) = :OLD.team;
    END IF;

    IF :NEW.team IS NOT NULL THEN
        UPDATE TeamTB t
            SET t.numOrder = t.numOrder + 1
```

```
            WHERE REF(t) = :NEW.team;
    END IF;
END;
```

CheckTeamInsertInitialization

Initializes numOrder and performanceScore in a correct way when inserting a new team (numOrder = 0, performanceScore = 1):

```
CREATE OR REPLACE TRIGGER CheckTeamInsertInitialization
BEFORE INSERT ON TeamTB
FOR EACH ROW
BEGIN
    :NEW.numOrder := 0;
    :NEW.performanceScore := 1;
END;
```

CheckNumEmployeeInTeam

Ensures a team cannot exceed 8 employees. This trigger makes use of a compound trigger to store the team reference and to avoid the mutating table problem:

```
CREATE OR REPLACE TRIGGER CheckNumEmployeeInTeam
FOR INSERT OR UPDATE OF team ON EmployeeTB
COMPOUND TRIGGER
    cnt number;
    teamRef REF TeamTY;
BEFORE EACH ROW IS
BEGIN
    teamRef := :New.team;
END BEFORE EACH ROW;

AFTER STATEMENT IS
BEGIN
    SELECT COUNT(*) INTO cnt FROM EmployeeTB e WHERE e.team = teamRef;
    IF cnt > 8 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Max number of employee
            reached');
    END IF;
    END AFTER STATEMENT;
END;
```

ComputePerformanceScore

Recalculates a team's average feedback score whenever a new order is added, updated or inserted. It stores the team affected by the operation, and compute the new score of them:

```
CREATE OR REPLACE TRIGGER ComputePerformanceScore
FOR INSERT OR UPDATE OR DELETE ON OrderTB
COMPOUND TRIGGER
    changedTeams TeamRefList := TeamRefList();

BEFORE EACH ROW IS
BEGIN
    IF DELETING OR UPDATING THEN
        IF :OLD.feedback IS NOT NULL AND :OLD.team IS NOT NULL THEN
            changedTeams.EXTEND;
            changedTeams(changedTeams.LAST) := :OLD.team;
        END IF;
    END IF;

    IF INSERTING OR UPDATING THEN
        IF :NEW.feedback IS NOT NULL AND :NEW.team IS NOT NULL THEN
            changedTeams.EXTEND;
            changedTeams(changedTeams.LAST) := :NEW.team;
        END IF;
    END IF;
END BEFORE EACH ROW;

AFTER STATEMENT IS
BEGIN
    UPDATE TeamTB t
        SET t.performanceScore = (
                SELECT ROUND(AVG(o.feedback.score), 2)
                    FROM OrderTB o
                WHERE o.team = REF(t)
                    AND o.feedback IS NOT NULL
            )
        WHERE REF(t) IN (
                SELECT COLUMN_VALUE
                    FROM TABLE(changedTeams)
            );
END AFTER STATEMENT;

END;
```

CheckScore

Handle the dangling references not caught by `ComputePerformanceScore`:

```
CREATE OR REPLACE TRIGGER CheckScore
```

```
BEFORE INSERT OR UPDATE ON TeamTB
FOR EACH ROW
BEGIN
    IF :NEW.numOrder = 0 AND :NEW.performanceScore != 1 THEN
        :NEW.performanceScore := 1;
    END IF;
END;
```

AddAccount

Automatically creates a business account for every new customer:

```
CREATE OR REPLACE TRIGGER AddAccount
FOR INSERT ON CustomerTB
COMPOUND TRIGGER
    customer VARCHAR2(11);
BEFORE EACH ROW IS
BEGIN
    customer := :NEW.VAT;
END BEFORE EACH ROW;

AFTER STATEMENT IS
    v_code VARCHAR2(10);
    v_exists NUMBER;
BEGIN
    insert into BusinessAccountTB values (
        sys_guid(),
        sysdate,
        (SELECT REF(c) FROM CustomerTB c WHERE c.VAT = customer)
    );
END AFTER STATEMENT;
END;
```

DeleteTeamAfterOperationalCenter

Deletes teams that lose their operational center reference upon an operational center's deletion:

```
CREATE OR REPLACE TRIGGER DeleteTeamAfterOperationalCenter
AFTER DELETE ON OperationalCenterTB
BEGIN
    DELETE FROM TeamTB t
    WHERE DEREF(t.operationalCenter) IS NULL AND t.operationalCenter
       IS NOT NULL;
END;
```

UpdateEmployeeAfterTeam

Sets an employee's team reference to NULL if the team is deleted, and unassigned orders if they're not completed:

```
CREATE OR REPLACE TRIGGER UpdateEmployeeAfterTeam
AFTER DELETE ON TeamTB
BEGIN
    UPDATE EmployeeTB e
    SET e.team = NULL
    WHERE DEREF(e.team) IS NULL AND e.team IS NOT NULL;

    UPDATE OrderTB o
    SET o.team = NULL
    WHERE DEREF(o.team) IS NULL AND o.team IS NOT NULL AND o.
       completionDate IS NULL;
END;
```

DeleteAccountAfterCustomer

Deletes business accounts that become orphaned when their customer is removed:

```
CREATE OR REPLACE TRIGGER DeleteAccountAfterCustomer
AFTER DELETE ON CustomerTB
BEGIN
    DELETE FROM BusinessAccountTB ba
    WHERE DEREF(ba.customer) IS NULL AND ba.customer IS NOT NULL;
END;
```

DeleteOrdersAfterTeam

Removes orders that have lost their team and business account references:

```
CREATE OR REPLACE TRIGGER DeleteOrdersAfterTeam
AFTER DELETE ON TeamTB
BEGIN
    DELETE FROM OrderTB o
    WHERE DEREF(o.team) IS NULL AND o.team IS NOT NULL AND DEREF(o.
       businessAccount) IS NULL AND o.businessAccount IS NOT NULL;
END;
```

DeleteOrdersAfterAccount

Deletes orders that have lost their related business account, under certain conditions:

- The business account reference is null and the completion date is null.
- The business account reference is null and the team reference is null.

In either case, there is no possibility of having useful information to compute the performance score, and there is no need to retain the order history because the order is not associated with any customer.

```
CREATE OR REPLACE TRIGGER DeleteOrdersAfterAccount
AFTER DELETE ON BusinessAccountTB
BEGIN
    DELETE FROM OrderTB o
    WHERE DEREF(o.businessAccount) IS NULL AND o.businessAccount IS
       NOT NULL AND o.completionDate IS NULL;

    DELETE FROM OrderTB o
    WHERE DEREF(o.businessAccount) IS NULL AND o.businessAccount IS
       NOT NULL AND (o.team IS NOT NULL AND DEREF(o.team) IS NULL);
END;
```

PreventOrderDeletion

Prevents order deletion unless it has lost all references or is uncompleted with no account:

```
CREATE OR REPLACE TRIGGER PreventOrderDeletion
BEFORE DELETE ON OrderTB
FOR EACH ROW
DECLARE
    v_team TeamTY;
    v_account BusinessAccountTY;
BEGIN
    IF :OLD.team IS NOT NULL THEN
        SELECT DEREF(:OLD.team) INTO v_team FROM DUAL;
    END IF;
    IF :OLD.businessAccount IS NOT NULL THEN
        SELECT DEREF(:OLD.businessAccount) INTO v_account FROM DUAL;
    END IF;

    IF NOT (
        (:OLD.team IS NOT NULL AND v_team IS NULL AND
            :OLD.businessAccount IS NOT NULL AND v_account IS NULL)
        OR
        (:OLD.businessAccount IS NOT NULL AND v_account IS NULL AND
            :OLD.completionDate IS NULL)
    ) THEN
        RAISE_APPLICATION_ERROR(-20019, 'Order deletion not allowed in
            this case');
    END IF;
END;
```

EmptyEmployeeListAfterTeamUpdate

Clears the employee list if a team reference changes:

```
CREATE OR REPLACE TRIGGER EmptyEmployeeListAfterTeamUpdate
BEFORE UPDATE OF team ON OrderTB
FOR EACH ROW
BEGIN
    IF :OLD.team IS NOT NULL AND :NEW.team IS NOT NULL AND :OLD.team
       != :NEW.team THEN
        :NEW.employees := NULL;
    END IF;
END;
```

6 | Database Population

populateCustomer

Populates the CustomerTB with random individual and business customers.

```
CREATE OR REPLACE PROCEDURE populateCustomer(individualCount IN NUMBER
    , businessCount IN NUMBER) AS
     maxC NUMBER;
     cc VARCHAR2(11);
BEGIN
     -- Get the maximum VAT number
     SELECT NVL(MAX(TO_NUMBER(SUBSTR(c.VAT, 3))), 0) INTO maxC FROM
        CustomerTB c;

     FOR i in 1..individualCount LOOP
          cc := LPAD(TO_CHAR(maxC + i), 9, '0');
          INSERT INTO CustomerTB VALUES (
               'IT' || TO_CHAR(cc),
               '+39' || TO_CHAR(DBMS_RANDOM.value(3000000000, 3999999999)
                  , 'FM0000000000'),
              DBMS_RANDOM.STRING('U', 10) || '@gmail.com',
               'individual',
               'customer' || DBMS_RANDOM.STRING('U', 10),
               'surname' || DBMS_RANDOM.STRING('U', 10),
              NULL,
              NULL,
              NULL
          );
     END LOOP;

     SELECT NVL(MAX(TO_NUMBER(SUBSTR(c.VAT, 3))), 0) INTO maxC FROM
        CustomerTB c;

     FOR i in 1..businessCount LOOP
          cc := LPAD(TO_CHAR(maxC + i), 9, '0');
          INSERT INTO CustomerTB VALUES (
```

```
            'IT' || TO_CHAR(cc),
            '+39' || TO_CHAR(DBMS_RANDOM.value(3000000000, 3999999999)
                , 'FM0000000000'),
            DBMS_RANDOM.STRING('U', 10) || '@gmail.com',
            'business',
            NULL,
            NULL,
            NULL,
            DBMS_RANDOM.STRING('U', 10),
            AddressTY(
                DBMS_RANDOM.STRING('U', 10),
                DBMS_RANDOM.value(1, 25),
                DBMS_RANDOM.STRING('U', 10),
                DBMS_RANDOM.STRING('U', 10),
                DBMS_RANDOM.STRING('U', 10),
                DBMS_RANDOM.STRING('U', 10)
            )
        );
    END LOOP;
END;
```

populateOperationalCenter

Creates multiple operational centers stored in OperationalCenterTB.

```
CREATE OR REPLACE PROCEDURE populateOperationalCenter(centerCount IN
   NUMBER) AS
BEGIN
    FOR i in 1..centerCount LOOP
        INSERT INTO OperationalCenterTB VALUES (
            'center-' || DBMS_RANDOM.STRING('U', 10),
            AddressTY(
                DBMS_RANDOM.STRING('U', 10),
                DBMS_RANDOM.value(1, 25),
                DBMS_RANDOM.STRING('U', 10),
                DBMS_RANDOM.STRING('U', 10),
                DBMS_RANDOM.STRING('U', 10),
                DBMS_RANDOM.STRING('U', 10)
            )
        );
    END LOOP;
END;
```

populateTeam

Generates random TeamTB entries, each assigned to a random operational center.

```
CREATE OR REPLACE PROCEDURE populateTeam(teamCount IN NUMBER) AS
BEGIN
    FOR i in 1..teamCount LOOP
        INSERT INTO TeamTB VALUES (
            RAWTOHEX(SYS_GUID()),
            'team-' || DBMS_RANDOM.STRING('U', 10),
            0,
            0,
            (SELECT * FROM (SELECT REF(o) FROM OperationalCenterTB o
               ORDER BY dbms_random.value()) FETCH FIRST 1 ROW ONLY)
        );
    END LOOP;
END;
```

populateEmployee

Inserts a specified number of employees into EmployeeTB, randomly associated with teams.

```
    CREATE OR REPLACE PROCEDURE populateEmployee(employeeCount IN
       NUMBER) AS
    TYPE refTeamTB IS TABLE OF REF TeamTY INDEX BY PLS_INTEGER;
    teams refTeamTB;
    currentTeamIndex PLS_INTEGER := 1;
    employeesInCurrentTeam NUMBER := 0;
    cc VARCHAR2(15);
BEGIN
    -- Bulk collect all teams
    SELECT REF(t) BULK COLLECT INTO teams
    FROM TeamTB t;

    FOR i in 1..employeeCount LOOP
        -- If current team has 7 employees, move to next team
        IF employeesInCurrentTeam = 7 THEN
            currentTeamIndex := currentTeamIndex + 1;
            employeesInCurrentTeam := 0;
            -- Exit if no more teams available
            IF currentTeamIndex > teams.COUNT THEN
                EXIT;
            END IF;
        END IF;

        INSERT INTO EmployeeTB VALUES (
```

```
            'E' || DBMS_RANDOM.STRING('U', 15),
            DBMS_RANDOM.STRING('U', 10),
            DBMS_RANDOM.STRING('U', 10),
            TO_DATE(
                TRUNC(DBMS_RANDOM.VALUE(TO_CHAR(DATE '1980-01-01', 'J'
                    ), TO_CHAR(DATE '1999-01-01', 'J'))),
                'J'
            ),
            '+39' || TO_CHAR(DBMS_RANDOM.value(3000000000, 3999999999)
                , 'FM0000000000'),
            DBMS_RANDOM.STRING('U', 10) || '@gmail.com',
            teams(currentTeamIndex)
        );
        employeesInCurrentTeam := employeesInCurrentTeam + 1;
    END LOOP;
END;
/
```

populateBusinessAccount

Fills BusinessAccountTB with random entries referencing existing customers.

```
CREATE OR REPLACE PROCEDURE populateBusinessAccount (numAccount IN
   NUMBER) AUTHID CURRENT_USER AS
    TYPE refCustomerTB IS TABLE OF REF CUSTOMERTY INDEX BY PLS_INTEGER
       ;
    customers refCustomerTB;
    randCustomer REF CUSTOMERTY;
    randIndex PLS_INTEGER;
BEGIN
    -- Fetch all customer refs into the collection
    SELECT REF(c) BULK COLLECT INTO customers FROM CustomerTB c;

    FOR i IN 1..numAccount LOOP
        randIndex := TRUNC(DBMS_RANDOM.VALUE(customers.FIRST,
           customers.LAST));
        randCustomer := customers(randIndex);

        INSERT INTO BusinessAccountTB values
        (
            RAWTOHEX(SYS_GUID()),
            sysdate,
            randCustomer
        );
    END LOOP;
```

```
END;
```

populateOrder

Inserts new orders into OrderTB, optionally assigning them randomly to teams.

```
CREATE OR REPLACE PROCEDURE populateOrder(orderCount IN NUMBER,
    probability IN NUMBER) AS
    TYPE refBusinessAccountTB IS TABLE OF REF BusinessAccountTY INDEX
        BY PLS_INTEGER;
    businessAccounts refBusinessAccountTB;
    randBA REF BusinessAccountTY;

    TYPE refTeamTB IS TABLE OF REF TeamTY INDEX BY PLS_INTEGER;
    teams refTeamTB;
    randTeam REF TeamTY;

    randIndexB PLS_INTEGER;
    randIndexT PLS_INTEGER;

BEGIN
    SELECT REF(b) BULK COLLECT INTO businessAccounts FROM
        BusinessAccountTB b;

    SELECT REF(t) BULK COLLECT INTO teams FROM TeamTB t;

    FOR i in 1..orderCount LOOP
        -- Probability of having a team
        randIndexB := TRUNC(DBMS_RANDOM.VALUE(businessAccounts.FIRST,
            businessAccounts.LAST));
        randBA := businessAccounts(randIndexB);
        IF DBMS_RANDOM.VALUE(0, 1) <= probability THEN
            randIndexT := TRUNC(DBMS_RANDOM.VALUE(teams.FIRST, teams.
                LAST));
            randTeam := teams(randIndexT);
            INSERT INTO OrderTB VALUES (
                RAWTOHEX(SYS_GUID()),
                TO_DATE(
                    TRUNC(DBMS_RANDOM.VALUE(TO_CHAR(DATE '2010-01-01',
                        'J'), TO_CHAR(DATE '2020-12-31', 'J'))), 'J'),
                CASE
                    WHEN DBMS_RANDOM.VALUE(0, 1) < 0.33 THEN 'online'
                    WHEN DBMS_RANDOM.VALUE(0, 1) < 0.66 THEN 'phone'
                    ELSE 'email'
                END,
```

```
                CASE
                    WHEN DBMS_RANDOM.VALUE(0, 1) < 0.33 THEN 'regular'
                    WHEN DBMS_RANDOM.VALUE(0, 1) < 0.66 THEN 'urgent'
                    ELSE 'bulk'
                END,
                DBMS_RANDOM.VALUE(1, 1000),
                randBA,
                randTeam,
                NULL,
                NULL,
                NULL
            );
        ELSE
            INSERT INTO OrderTB VALUES (
                RAWTOHEX(SYS_GUID()),
                TO_DATE(
                    TRUNC(DBMS_RANDOM.VALUE(TO_CHAR(DATE '2010-01-01',
                        'J'), TO_CHAR(DATE '2020-12-31', 'J'))), 'J'),
                CASE
                    WHEN DBMS_RANDOM.VALUE(0, 1) < 0.33 THEN 'online'
                    WHEN DBMS_RANDOM.VALUE(0, 1) < 0.66 THEN 'phone'
                    ELSE 'email'
                END,
                CASE
                    WHEN DBMS_RANDOM.VALUE(0, 1) < 0.33 THEN 'regular'
                    WHEN DBMS_RANDOM.VALUE(0, 1) < 0.66 THEN 'urgent'
                    ELSE 'bulk'
                END,
                DBMS_RANDOM.VALUE(1, 1000),
                randBA,
                NULL,
                NULL,
                NULL,
                NULL
            );
        END IF;
    END LOOP;
END;
```

populateEmployeeInOrder

Randomly assigns employees to orders that are incomplete and already linked to a team.

```
CREATE OR REPLACE PROCEDURE populateEmployeeInOrder(probability IN
    NUMBER) AS
```

```
BEGIN
    FOR orderRow IN (SELECT o.ID, DEREF(o.team) AS team FROM OrderTB o
        WHERE o.team IS NOT NULL AND o.completionDate IS NULL AND o.
       feedback IS NULL) LOOP
        IF DBMS_RANDOM.VALUE(0, 1) <= probability THEN
            -- Create temporary varray
            DECLARE
                emp_array EmployeeVA := EmployeeVA();
                v_team_emp_count NUMBER;
            BEGIN
                -- Get number of employees in the team
                SELECT COUNT(*) INTO v_team_emp_count
                FROM EmployeeTB e
                WHERE DEREF(e.team).ID = orderRow.team.ID;

                -- Get random number of employees (0 to team size)
                   from the same team
                FOR i in 1..DBMS_RANDOM.value(0, v_team_emp_count)
                   LOOP
                    -- Extend the varray
                    emp_array.EXTEND;
                    -- Get random employee reference from the same
                       team
                    SELECT REF(e) INTO emp_array(emp_array.COUNT)
                    FROM EmployeeTB e
                    WHERE DEREF(e.team).ID = orderRow.team.ID
                    ORDER BY dbms_random.value()
                    FETCH FIRST 1 ROW ONLY;
                END LOOP;

                -- Update the order with the new employee array
                UPDATE OrderTB o
                SET o.employees = emp_array
                WHERE o.ID = orderRow.ID;
            END;
        END IF;
    END LOOP;
END;
```

populateCompletionDateAndFeedbackInOrder

Optionally sets the completion date and feedback for orders that still lack both.

```
CREATE OR REPLACE PROCEDURE populateCompletionDateAndFeedbackInOrder(
   probability IN NUMBER) AS
```

```
    v_team_id VARCHAR2(50);
    v_team_name VARCHAR2(50);
    v_team_score NUMBER;
BEGIN
    FOR orderRow IN (SELECT o.ID FROM OrderTB o WHERE o.team IS NOT
       NULL AND o.completionDate IS NULL AND O.feedback IS NULL) LOOP
        -- Only process orders based on probability
        IF DBMS_RANDOM.VALUE(0, 1) <= probability THEN
            -- Update the order with the new completion date and
               feedback
            UPDATE OrderTB o
            SET o.completionDate = TO_DATE(
                TRUNC(DBMS_RANDOM.VALUE(TO_CHAR(DATE '2021-01-01', 'J'
                   ), TO_CHAR(DATE '2021-12-31', 'J'))),
                'J'
            ),
            o.feedback = FeedbackTY(
                ROUND(DBMS_RANDOM.VALUE(1, 5)),
                CASE
                    WHEN DBMS_RANDOM.VALUE(0, 1) < 0.33 THEN 'Great
                       service!'
                    WHEN DBMS_RANDOM.VALUE(0, 1) < 0.66 THEN 'Could be
                        better'
                    ELSE 'Average experience'
                END
            )
            WHERE o.ID = orderRow.ID;

            SELECT t.ID, t.name, t.performanceScore
            INTO v_team_id, v_team_name, v_team_score
            FROM TeamTB t
            WHERE t.ID = (SELECT DEREF(o.team).ID FROM OrderTB o WHERE
                o.ID = orderRow.ID);
        END IF;
    END LOOP;
END;
```

7 | Procedures and Functions

Operation 1: registerCustomer

Registers a new customer with optional business or individual data.

```
CREATE OR REPLACE PROCEDURE registerCustomer(
    VAT IN VARCHAR2,
    phone IN VARCHAR2,
    email IN VARCHAR2,
    type IN VARCHAR2,
    name IN VARCHAR2,
    surname IN VARCHAR2,
    dob IN DATE,
    companyName IN VARCHAR2,
    address IN AddressTY
) AS
    customer CustomerTY;
BEGIN
    IF type = 'individual' THEN
        IF VAT IS NULL THEN
            customer := CustomerTY(SYS_GUID(), phone, email, type,
                name, surname, dob, NULL, NULL);
        ELSE
            customer := CustomerTY(VAT, phone, email, type, name,
                surname, dob, NULL, NULL);
        END IF;
    ELSIF type = 'business' THEN
        IF VAT IS NULL THEN
            customer := CustomerTY(SYS_GUID(), phone, email, type,
                NULL, NULL, NULL, companyName, address);
        ELSE
            customer := CustomerTY(VAT, phone, email, type, NULL, NULL
                , NULL, companyName, address);
        END IF;
    ELSE
        RAISE_APPLICATION_ERROR(-20000, 'Invalid customer type; must
```

```
            be either individual or business');
    END IF;
    INSERT INTO CustomerTB VALUES customer;
    COMMIT;
END;
```

Operation 2: addOrder

Adds a new order by referencing a business account and optional employees.

```
CREATE OR REPLACE PROCEDURE addOrder(
    ID IN VARCHAR2,
    placingDate IN DATE,
    orderMode IN VARCHAR2,
    orderType IN VARCHAR2,
    cost IN NUMBER,
    businessAccountName IN VARCHAR2,
    employees IN EmployeeVA DEFAULT NULL
) AS
    baRef REF BusinessAccountTY;
BEGIN
    SELECT REF(b)
    INTO baRef
    FROM BusinessAccountTB b
    WHERE b.CODE = businessAccountName;
    IF ID IS NULL THEN
        INSERT INTO OrderTB (ID, placingDate, orderMode, orderType,
            cost, businessAccount, employees) VALUES (sys_GUID(),
            placingDate, orderMode, orderType, cost, baRef, employees);
    ELSE
        INSERT INTO OrderTB (ID, placingDate, orderMode, orderType,
            cost, businessAccount, employees) VALUES (ID, placingDate,
            orderMode, orderType, cost, baRef, employees);
    END IF;
    COMMIT;
END;
```

Operation 3: assignOrderToTeam

Assigns an order to a specific team.

```
CREATE OR REPLACE PROCEDURE assignOrderToTeam(
    orderID IN VARCHAR2,
    teamID IN VARCHAR2
) AS
BEGIN
```

```
    -- Update the order with the team reference
    UPDATE OrderTB
    SET team = (SELECT REF(t) FROM TeamTB t WHERE t.ID = teamID)
    WHERE ID = orderID;
    COMMIT;
END;
```

Operation 4A: totalNumOrder

Returns the total number of orders handled by a given team.

```
CREATE OR REPLACE FUNCTION totalNumOrder(
    teamID IN VARCHAR2
) RETURN NUMBER AS
    v_count NUMBER;
BEGIN
    SELECT numOrder INTO v_count
    FROM TeamTB
    WHERE ID = teamID;
    RETURN NVL(v_count, 0);
END;
```

Operation 4B: totalOrderCost

Calculates the total cost of all orders for a team.

```
CREATE OR REPLACE FUNCTION totalOrderCost(
    teamID IN VARCHAR2
) RETURN NUMBER AS
    totalCost NUMBER;
BEGIN
    SELECT SUM(cost) INTO totalCost
    FROM OrderTB o
    WHERE o.team.ID = teamID;
    RETURN NVL(totalCost, 0);
END;
```

Operation 5: printTeamsByPerformanceScore

Prints teams sorted by performance score in descending order.

```
CREATE OR REPLACE PROCEDURE printTeamsByPerformanceScore AS
BEGIN
    FOR team IN (SELECT * FROM TeamTB ORDER BY performanceScore DESC)
       LOOP
        DBMS_OUTPUT.PUT_LINE('Team ID: ' || team.ID || ', Performance
            Score: ' || team.performanceScore);
```

```
    END LOOP;
    COMMIT;
END;
```

8 | Physical Design

In this chapter we will study the `EXPLAIN PLAN` of the operations implemented to see if there is the need to define new indexes.

Operation 1 (10 times per day)

```
PLAN_TABLE_OUTPUT
1
2  ----------------------------------------------------------------------------------------
3  | Id  | Operation                | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
4  ----------------------------------------------------------------------------------------
5  |   0 | INSERT STATEMENT         |            |     1 |   100 |     1   (0)| 00:00:01 |
6  |   1 |  LOAD TABLE CONVENTIONAL | CUSTOMERTB |       |       |            |          |
7  ----------------------------------------------------------------------------------------
```

Considerations

The first query is a simple insert, so we don't need to optimize anything.

Operation 2 (1000 times per day)

Operation 2 is composed of 2 queries, the first that retrieve the business account and the second that add the order.

First query

```
PLAN_TABLE_OUTPUT
1  Plan hash value: 1254032843
2
3  ----------------------------------------------------------------------------------------------------
4  | Id  | Operation                    | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
5  ----------------------------------------------------------------------------------------------------
6  |   0 | SELECT STATEMENT             |                    |     1 |    29 |     1   (0)| 00:00:01 |
7  |   1 |  TABLE ACCESS BY INDEX ROWID| BUSINESSACCOUNTTB |     1 |    29 |     1   (0)| 00:00:01 |
8  |*  2 |   INDEX UNIQUE SCAN          | SYS_C0024322       |     1 |       |     1   (0)| 00:00:01 |
9  ----------------------------------------------------------------------------------------------------
10
11 Predicate Information (identified by operation id):
12 ---------------------------------------------------
13
14    2 - access("B"."CODE"='B000000001')
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	1
TABLE ACCESS	BUSINESSACCOUNTTB	BY INDEX ROWID	1	1
INDEX	SYS_C0024322	UNIQUE SCAN	1	1
Access Predicates				
B.CODE='B000000001'				

Second query

```
PLAN_TABLE_OUTPUT
Plan hash value: 1254032843

-----------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | INSERT STATEMENT              |                    |     1 |   100 |     1   (0)| 00:00:01 |
|   1 |  LOAD TABLE CONVENTIONAL      | ORDERTB            |       |       |            |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| BUSINESSACCOUNTTB  |     1 |    29 |     1   (0)| 00:00:01 |
|*  3 |    INDEX UNIQUE SCAN          | SYS_C0024322       |     1 |       |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

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

   3 - access("B"."CODE"='B000000001')
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
INSERT STATEMENT			1	1
LOAD TABLE CONVENTIONAL	ORDERTB			
TABLE ACCESS	BUSINESSACCOUNTTB	BY INDEX ROWID	1	1
INDEX	SYS_C0024322	UNIQUE SCAN	1	1
Access Predicates				
B.CODE='B000000001'				

Considerations

The first query retrieves the business account from a key value, so it's already indexed. The second query is an insert, so we don't need to optimize anything.

Operation 3 (500 times per day)

```
PLAN_TABLE_OUTPUT
Plan hash value: 3482816316

-----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | UPDATE STATEMENT              |              |     1 |    34 |     3  (34)| 00:00:01 |
|   1 |  UPDATE                       | ORDERTB      |       |       |            |          |
|*  2 |   INDEX UNIQUE SCAN           | SYS_C0024335 |     1 |    34 |     1   (0)| 00:00:01 |
|   3 |   TABLE ACCESS BY INDEX ROWID| TEAMTB       |     1 |    49 |     1   (0)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN          | SYS_C0024302 |     1 |       |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

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

   2 - access("O"."ID"='O000001111')
   4 - access("T"."ID"='T000001')
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
UPDATE STATEMENT			1	3
UPDATE	ORDERTB			
INDEX	SYS_C0024335	UNIQUE SCAN	1	1
Access Predicates				
O.ID='O000001111'				
TABLE ACCESS	TEAMTB	BY INDEX ROWID	1	1
INDEX	SYS_C0024302	UNIQUE SCAN	1	1
Access Predicates				
T.ID='T000001'				

Considerations

The query is a simple update with punctual selection on a key value, so we don't need to optimize anything.

Operation 4A (200 times per day)

```
PLAN_TABLE_OUTPUT
Plan hash value: 2943281028

-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |              |     1 |    40 |     1   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| TEAMTB       |     1 |    40 |     1   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN          | SYS_C0024302 |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   2 - access("ID"='2C6535D4B8FF9D06E063020012AC4D2C')
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	1
TABLE ACCESS	TEAMTB	BY INDEX ROWID	1	1
INDEX	SYS_C0024302	UNIQUE SCAN	1	1
Access Predicates				
ID='2C6535D4B8FF9D06E063020012AC4D2C'				

Considerations

The query is a select with a punctual selection on a key value, so we don't need to optimize anything.

Operation 4B (200 times per day)

```
PLAN_TABLE_OUTPUT
Plan hash value: 2930154078

--------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |     1 |    40 |     2   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE    |         |     1 |    40 |            |          |
|*  2 |   TABLE ACCESS FULL| ORDERTB |     1 |    40 |     2   (0)| 00:00:01 |
--------------------------------------------------------------------------------

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

   2 - filter("O"."TEAM"."ID"='2C6535D4B8FF9D06E063020012AC4D2C')
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	2
SORT		AGGREGATE	1	
TABLE ACCESS	ORDERTB	FULL	1	2
Filter Predicates				
O.TEAM.ID='2C6535D4B8FF9D06E063020012AC4D2C'				

Considerations

The query is a select with a punctual selection on a key value, with a full table scan given by the *sum* function, but given the selection of a specific team on the basis of its key value, we don't need to optimize anything.

Operation 5 (20 times per day)

```
PLAN_TABLE_OUTPUT
Plan hash value: 4254525359

-------------------------------------------------------------------------------
| Id  | Operation          | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |        |     1 |    92 |     3  (34)| 00:00:01 |
|   1 |  SORT ORDER BY     |        |     1 |    92 |     3  (34)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| TEAMTB |     1 |    92 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	3
SORT		ORDER BY	1	3
TABLE ACCESS	TEAMTB	FULL	1	2

Considerations

The query needs to scan all the table `TeamTB` anyway, and `performanceScore` has duplicates, so we don't optimize.

9 | Web Application

This chapter presents a simple Flask application connecting to the Oracle database. It employs multiple routes for the main operations:

- **Order management.** Users can insert new orders (`/add_order`) and assign them to teams (`/assign_order`).
- **Customer registration.** A new customer can be added (`/register_customer`), supporting both ”individual” and ”business” types.
- **Team statistics.** Routes like `/team_stats` and `/teams_list` give information about total orders handled by a specific team, the total cost of the orders handled by a specific team and information about performance score of all teams.

The application follows this directory structure:

```
webapp/
 database.py
 app.py
 static/
    css/
        style.css
 templates/
     base.html
     home.html
     add_order.html
     assign_order.html
     register_customer.html
     team_stats.html
     teams_list.html
```

The code centralizes database access in a separate `database.py` module. In production this file will be replaced placing the database credentials in an `env`, accessed with the library `python-dotenv`.

Procedure and function calls (e.g., `registerCustomer`, `addOrder`, or `assignOrderToTeam`) are used to interact with the Oracle schema, ensuring logic remains in the PL/SQL layer.

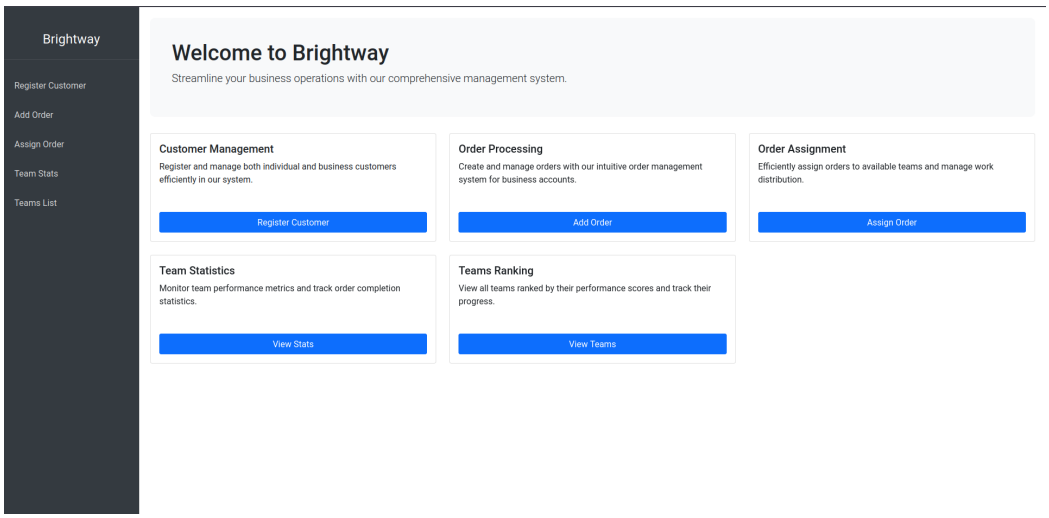


Figure 9.1: Home page of the web application.

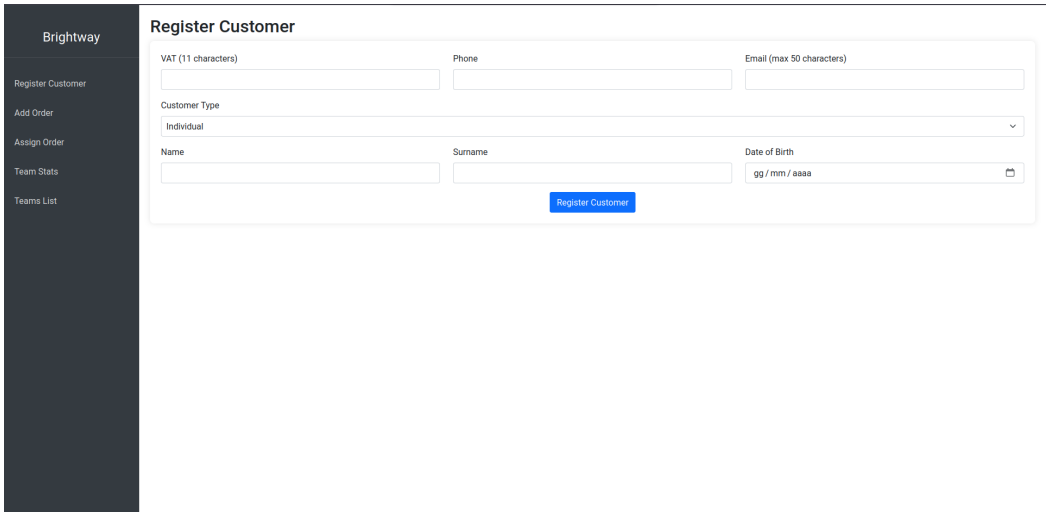


Figure 9.2: Customer registration form for individual customers.

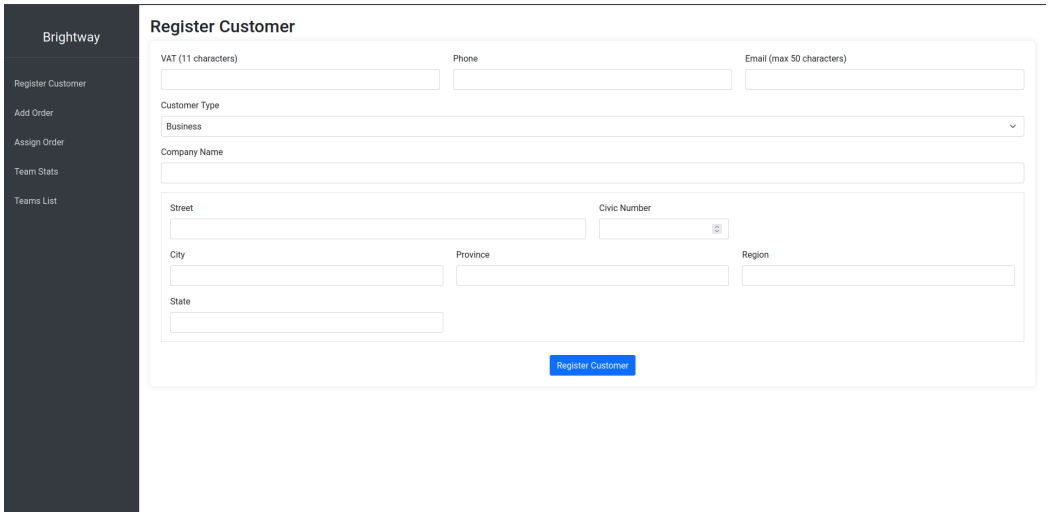


Figure 9.3: Customer registration form for business customers.

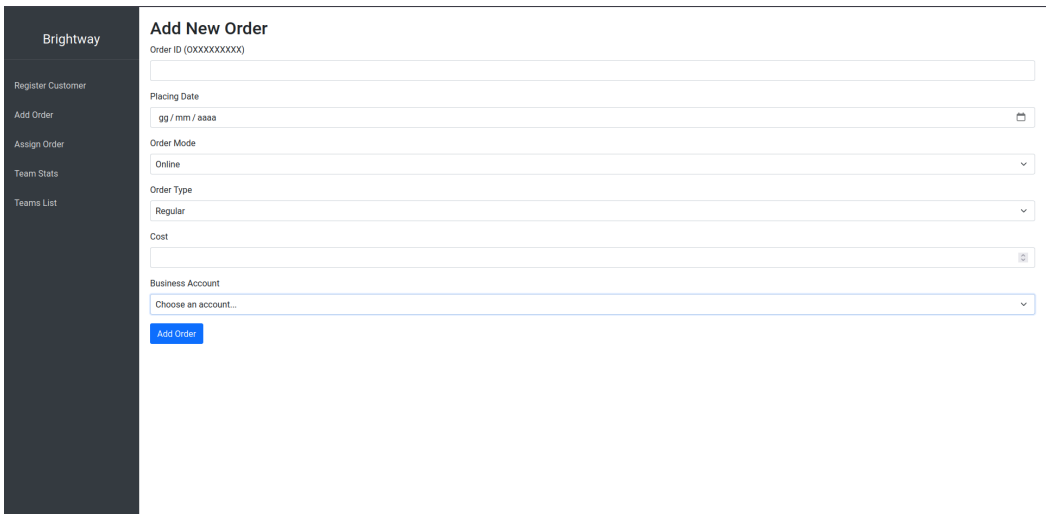


Figure 9.4: Order insertion form.

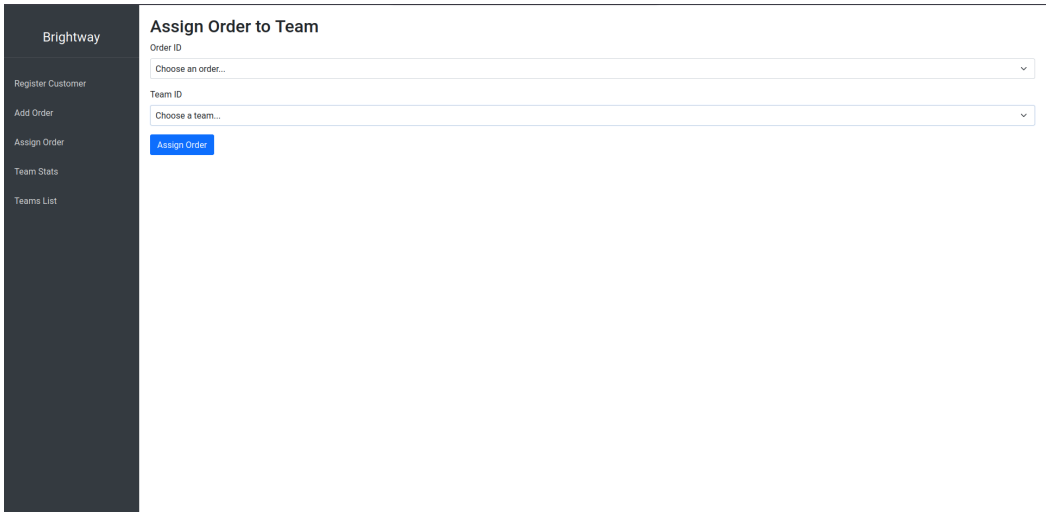


Figure 9.5: Order assignment form.

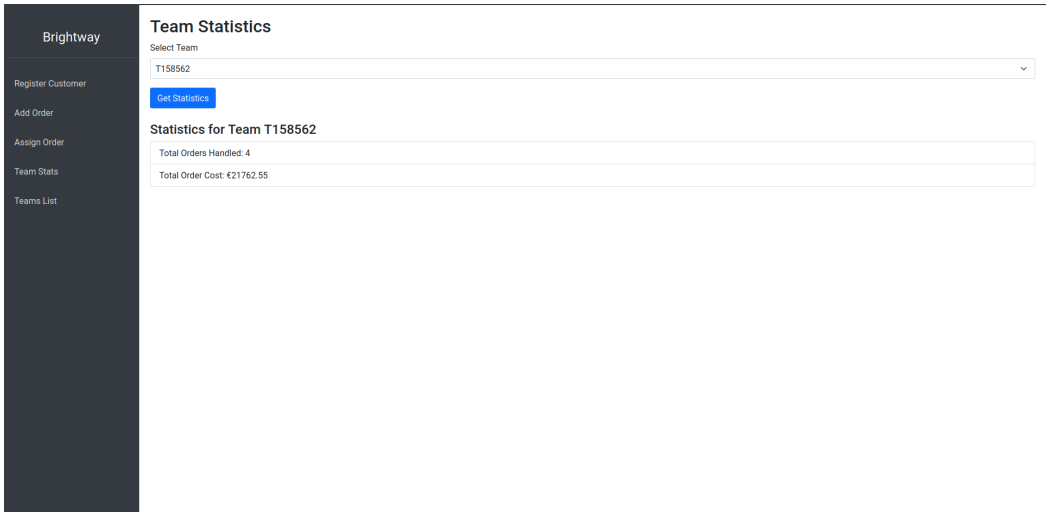


Figure 9.6: Team statistics page.

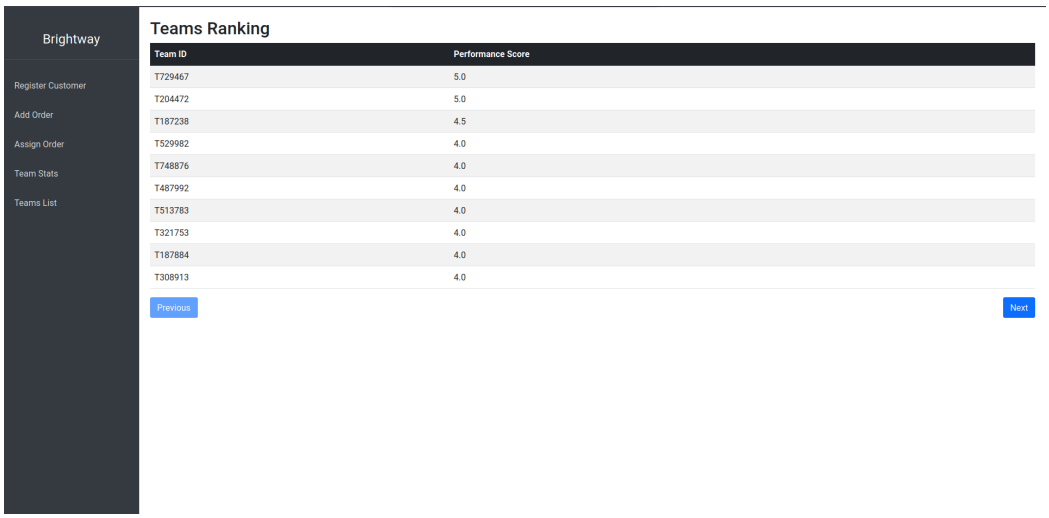


Figure 9.7: List of teams sorted by performance score.

Here we present only the code of `app.py` for brevity:

```
from flask import Flask, render_template, request, redirect, url_for
from datetime import datetime
import database
import oracledb

app = Flask(__name__)

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

# Op (1)
@app.route('/register_customer', methods=['GET', 'POST'])
def register_customer():
    if request.method == 'POST':
        try:
            conn = database.get_connection()
            cursor = conn.cursor()

            vat = request.form['vat']
            phone = request.form['phone']
            email = request.form['email']
            cust_type = request.form['type']

            # Always create an AddressTY object (even if empty for
               individual)
            address_type = conn.gettype('ADDRESSTY')
            address_obj = address_type.newobject()
```

```
            address_obj.STREET = None
            address_obj.CIVICNUM = None
            address_obj.CITY = None
            address_obj.PROVINCE = None
            address_obj.REGION = None
            address_obj.STATE = None

            if cust_type == 'individual':
                name = request.form['name']
                surname = request.form['surname']
                dob = datetime.strptime(request.form['dob'], '%Y-%m-%d
                    ').date()
                company_name = None
            else:
                name = None
                surname = None
                dob = None
                company_name = request.form['companyName']
                address_obj.STREET = request.form['street']
                address_obj.CIVICNUM = int(request.form['civicNum'])
                address_obj.CITY = request.form['city']
                address_obj.PROVINCE = request.form['province']
                address_obj.REGION = request.form['region']
                address_obj.STATE = request.form['state']

            cursor.callproc('registerCustomer', [
                vat, phone, email, cust_type,
                name, surname, dob,
                company_name, address_obj
            ])

            conn.commit()
            cursor.execute("""
                SELECT CODE
                FROM BusinessAccountTB b
                WHERE b.customer.VAT = :vat
            """, [vat])
            result = cursor.fetchone()
            if result:
                return render_template('register_customer.html',
                    business_account=result[0])
            return render_template('register_customer.html')

        except oracledb.DatabaseError as e:
```

```
            error, = e.args
            return render_template('register_customer.html', error=
                error.message)
        finally:
            cursor.close()
            conn.close()

    return render_template('register_customer.html')

# Op. (2)
@app.route('/add_order', methods=['GET', 'POST'])
def add_order():
    if request.method == 'POST':
        try:
            conn = database.get_connection()
            cursor = conn.cursor()

            # Form data
            order_id = request.form['order_id']
            placing_date = datetime.strptime(request.form['
                placing_date'], '%Y-%m-%d').date()

            guid = False
            if order_id == "":
                # execute sys_guid() to get a unique identifier
                cursor.execute("SELECT RAWTOHEX(SYS_GUID()) FROM dual"
                    )
                order_id = cursor.fetchone()[0]
                guid = True

            cursor.callproc('addOrder', [
                order_id,
                placing_date,
                request.form['order_mode'],
                request.form['order_type'],
                float(request.form['cost']),
                request.form['business_account']
            ])

            conn.commit()
            # Get business accounts before rendering template
            cursor.execute("SELECT CODE FROM BusinessAccountTB")
            business_accounts = [row[0] for row in cursor.fetchall()]
            if guid:
```

```
                return render_template('add_order.html', success="
                    Order added successfully! Order ID: " + order_id,
                    business_accounts=business_accounts)
            return render_template('add_order.html', success="Order
                added successfully!", business_accounts=
                business_accounts)

        except oracledb.DatabaseError as e:
            error, = e.args
            # Get business accounts even when there's an error
            cursor.execute("SELECT CODE FROM BusinessAccountTB")
            business_accounts = [row[0] for row in cursor.fetchall()]
            return render_template('add_order.html', error=error.
                message, business_accounts=business_accounts)
        finally:
            cursor.close()
            conn.close()

    # Handle GET request
    try:
        conn = database.get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT CODE FROM BusinessAccountTB")
        business_accounts = [row[0] for row in cursor.fetchall()]
        return render_template('add_order.html', business_accounts=
            business_accounts)
    except oracledb.DatabaseError as e:
        error, = e.args
        return render_template('add_order.html', error=error.message)
    finally:
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()


# Op. (3)
@app.route('/assign_order', methods=['GET', 'POST'])
def assign_order():
    if request.method == 'POST':
        try:
            conn = database.get_connection()
            cursor = conn.cursor()
            cursor.callproc('assignOrderToTeam', [
```

```
                request.form['order_id'],
                request.form['team_id']
            ])
            conn.commit()
            # Get lists for dropdowns
            cursor.execute("SELECT ID FROM OrderTB WHERE team IS NULL"
               )
            orders = [row[0] for row in cursor.fetchall()]
            cursor.execute("SELECT ID FROM TeamTB")
            teams = [row[0] for row in cursor.fetchall()]
            return render_template('assign_order.html', orders=orders,
                teams=teams, success="Order assigned successfully!")
        except oracledb.DatabaseError as e:
            error, = e.args
            # Get lists for dropdowns even when there's an error
            cursor.execute("SELECT ID FROM OrderTB WHERE team IS NULL"
               )
            orders = [row[0] for row in cursor.fetchall()]
            cursor.execute("SELECT ID FROM TeamTB")
            teams = [row[0] for row in cursor.fetchall()]
            return render_template('assign_order.html', error=error.
               message, orders=orders, teams=teams)
        finally:
            cursor.close()
            conn.close()

    # Get lists for dropdowns
    try:
        conn = database.get_connection()
        cursor = conn.cursor()
        # Get unassigned orders
        cursor.execute("SELECT ID FROM OrderTB WHERE team IS NULL")
        orders = [row[0] for row in cursor.fetchall()]
        # Get all teams
        cursor.execute("SELECT ID FROM TeamTB")
        teams = [row[0] for row in cursor.fetchall()]
        return render_template('assign_order.html', orders=orders,
           teams=teams)
    except oracledb.DatabaseError as e:
        error, = e.args
        return render_template('assign_order.html', error=error.
           message)
    finally:
        cursor.close()
```

```
        conn.close()

# Op. (4A), (4B)
@app.route('/team_stats', methods=['GET', 'POST'])
def team_stats():
    if request.method == 'POST':
        team_id = request.form['team_id']
        try:
            conn = database.get_connection()
            cursor = conn.cursor()

            # Get total orders
            cursor.execute("SELECT totalNumOrder(:1) FROM DUAL", [
                team_id])
            total_orders = cursor.fetchone()[0]

            # Get total cost
            cursor.execute("SELECT totalOrderCost(:1) FROM DUAL", [
                team_id])
            total_cost = cursor.fetchone()[0]

            # Get teams for dropdown
            cursor.execute("SELECT ID FROM TeamTB")
            teams = [row[0] for row in cursor.fetchall()]

            return render_template('team_stats.html',
                                   total_orders=total_orders,
                                   total_cost=total_cost,
                                   team_id=team_id,
                                   teams=teams)
        except oracledb.DatabaseError as e:
            error, = e.args
            return render_template('team_stats.html', error=error.
                message)
        finally:
            cursor.close()
            conn.close()

    # For GET request, just get teams for dropdown
    try:
        conn = database.get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT ID FROM TeamTB")
        teams = [row[0] for row in cursor.fetchall()]
```

```
        return render_template('team_stats.html', teams=teams)
    except oracledb.DatabaseError as e:
        error, = e.args
        return render_template('team_stats.html', error=error.message)
    finally:
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()

# Op. (5)
@app.route('/teams_list', methods=['GET', 'POST'])
def teams_list():
    # Default number of rows to display
    rows_to_display = 10
    # Get the current offset from query parameters, default to 0
    offset = int(request.args.get('offset', 0))

    if offset < 0:
        offset = 0

    if request.method == 'POST':
        # Get number of rows from form input
        rows_to_display = int(request.form.get('num_rows', 20))
        # Reset offset when form is submitted
        offset = 0

    try:
        conn = database.get_connection()
        cursor = conn.cursor()

        # Get total count of teams
        cursor.execute("SELECT COUNT(*) FROM TeamTB")
        total_teams = cursor.fetchone()[0]

        # Modified query to include OFFSET
        cursor.execute("""
            SELECT ID, performanceScore
            FROM TeamTB
            ORDER BY performanceScore DESC
            OFFSET :1 ROWS FETCH NEXT :2 ROWS ONLY
        """, [offset, rows_to_display])
        teams = cursor.fetchall()
```

```
        # Check if there are more teams to show
        has_next = (offset + rows_to_display) < total_teams

        return render_template('teams_list.html',
                               teams=teams,
                               num_rows=rows_to_display,
                               offset=offset,
                               has_next=has_next)
    except oracledb.DatabaseError as e:
        error, = e.args
        return render_template('teams_list.html', error=error.message)
    finally:
        cursor.close()
        conn.close()

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