



UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

Computer Science Department

Master Degree in Computer Science

Practical Part of **Database Systems**

Brightway: Technical Documentation

Student:

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

Student ID:

832437

ACADEMIC YEAR 2024-2025

Contents

1	Requirements	5
2	Conceptual Design	7
3	Logical Design	13
4	Implementation	19
5	Trigger Implementation	23
6	Database Population	33
7	Procedures and Functions	41
8	Physical Design	45
9	Web Application	49

1 | Requirements

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

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*.

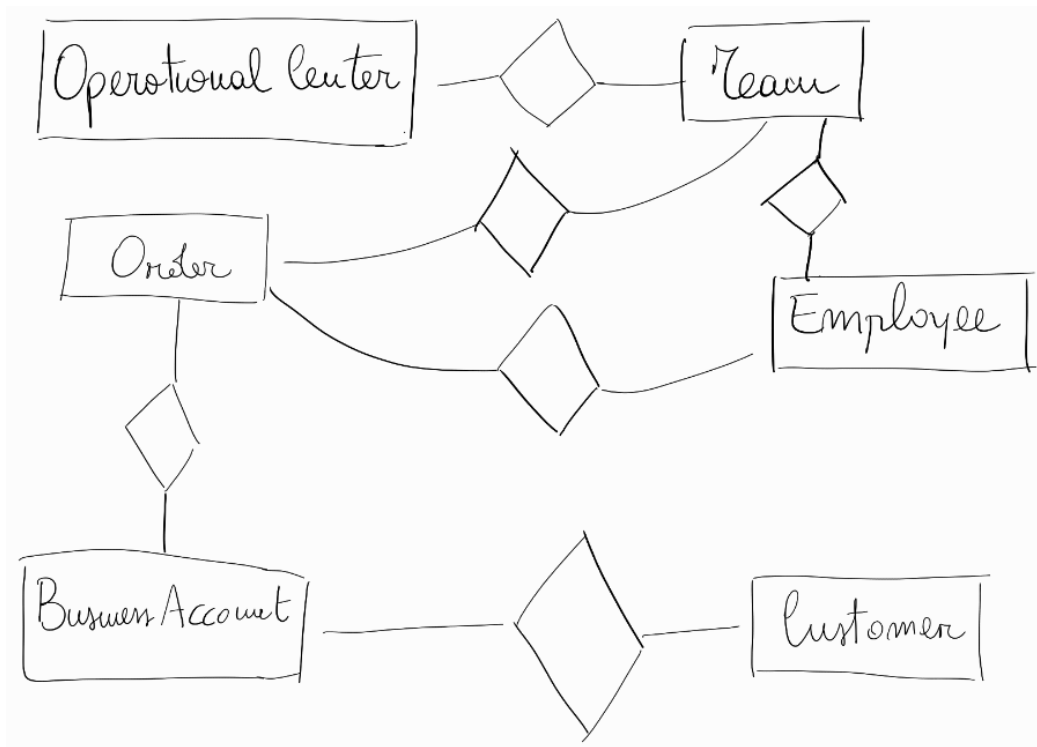
Number of operations is replaced by *number of orders*.

Glossary of terms

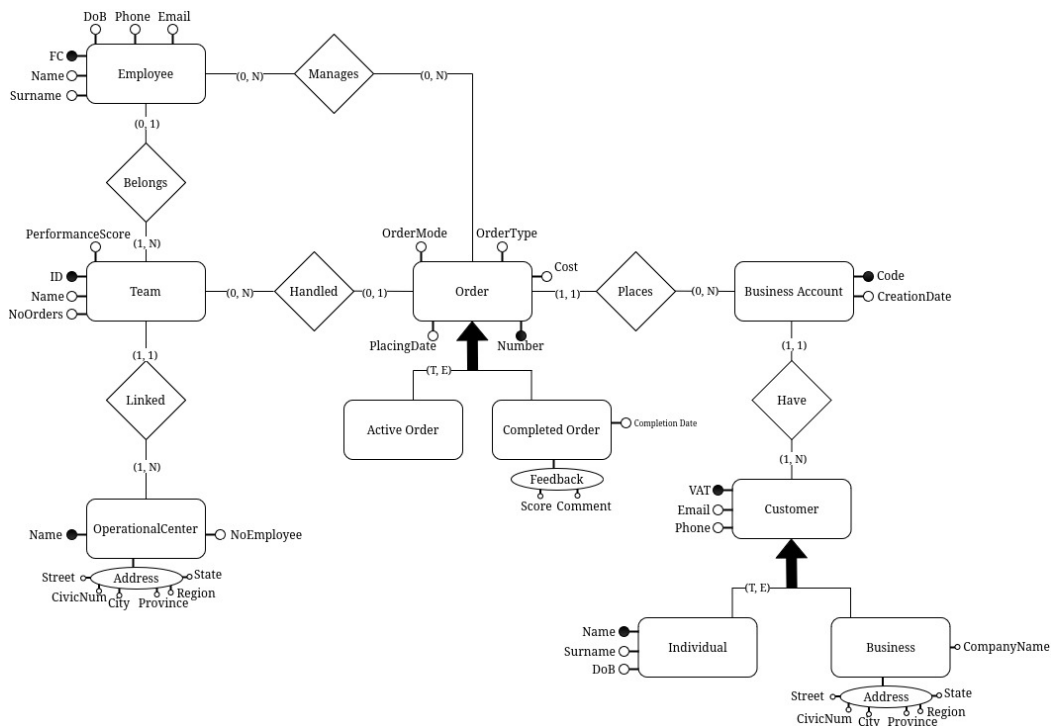
Term	Description	Synonyms	Connections
Operational Center	Decentralized locations handling local storage and shipments. Characterized by name, address, city/province, and number of employees.	/	Team
Order	Requests for services or goods from customers, identified by type (regular, urgent, or bulk). Includes 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 (<i>given</i>)	150
Handled By	R	45000 (same as (assigned) Order)	45000
Order	E	$300 \text{ op/m} \cdot 150 + 100 \text{ not assigned}$	45100
Belongs To	R	$150 \text{ Team} \cdot 6.4 \text{ current Employee}$	960
Employee	E	$80\% \text{ of } (150 \cdot 8) + 140 \text{ past Employee}$	1100
Manages	R	$45000 \cdot 3 \text{ (assuming 3 Employee working on an Order)}$	135000
Places	R	45100 (same as Order)	45100
Business Account	E	$\frac{45100 \text{ Order}}{1.5 \text{ Order/month}}$	30000
Have	R	30000 (same as Business Account)	30000
Customer	E	$\frac{30000 \text{ Business Account}}{1.5 \text{ Business Account/Customer}}$	20000
Active Order	E	$20\% \text{ of } 45000 + 100 \text{ not assigned}$	9100
Completed Order	E	$80\% \text{ of } 45000$	36000
Individual	E	$90\% \text{ of Customer}$	18000
Business	E	$10\% \text{ of Customer}$	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: $6 \text{ accesses} \cdot 10 = 60 \text{ accesses/day}$

Operation 2: Add a new order

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

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

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): $8 \text{ accesses} \cdot 500 = 4000 \text{ accesses/day}$

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

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): $300 \text{ accesses} \cdot 200 = 60000 \text{ accesses/day}$

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

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
Team	E	1	R
Handled	R	300	R
Order	E	300	R

Operation cost (*without redundancy*): $601 \text{ accesses} \cdot 200 = 60000 \text{ accesses/day}$

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*): $150 \text{ accesses} \cdot 20 = 3000 \text{ accesses/day}$

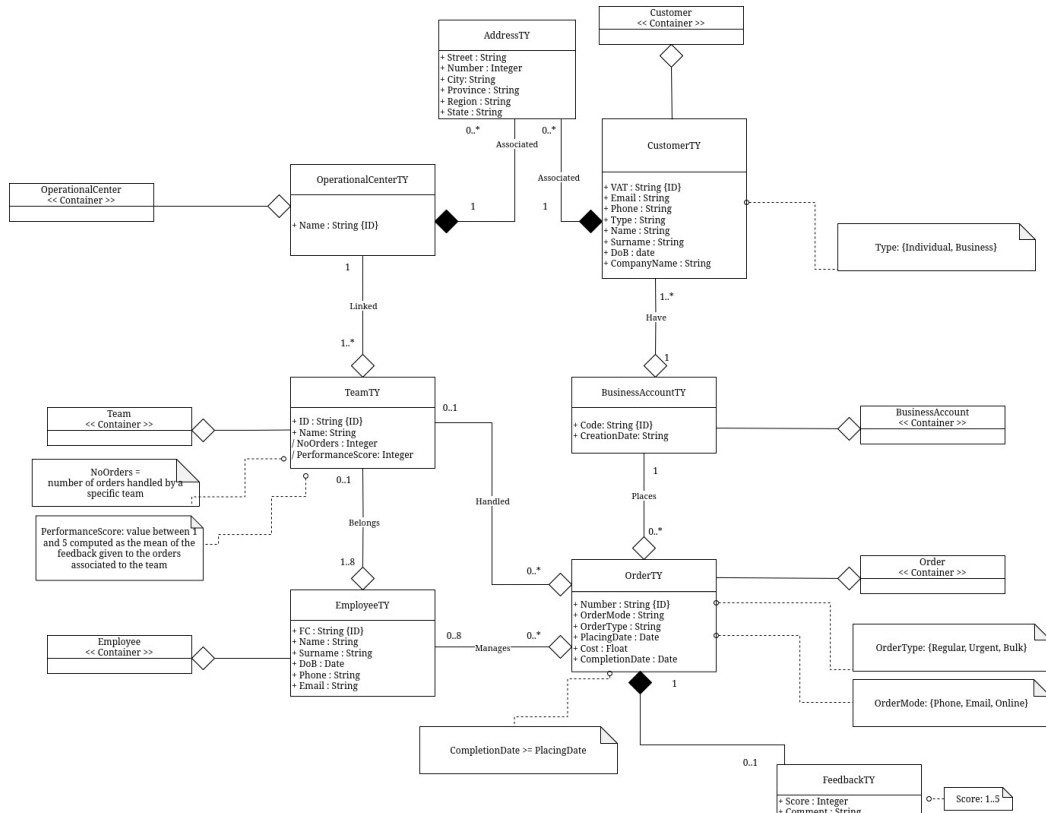
Redundancy Analysis

- 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 124,000 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
    — * FUNZIONA
    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;

    — * FUNZIONA
    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;

— * FUNZIONA

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;

— * FUNZIONA

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;

— * FUNZIONA

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;

— * FUNZIONA

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;

    — * FUNZIONA
    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 TYPE TeamRefList AS TABLE OF REF TeamTY;

CREATE OR REPLACE TRIGGER ComputePerformanceScore
FOR INSERT OR UPDATE OR DELETE OF feedback 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;
```

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 feedback score 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
    maxE NUMBER;
    TYPE refTeamTB IS TABLE OF REF TeamTY INDEX BY PLS_INTEGER;
    teams refTeamTB;
    currentTeamIndex PLS_INTEGER := 1;
    employeesInCurrentTeam NUMBER := 0;
    cc VARCHAR2(15);
BEGIN
    SELECT NVL(MAX(TO_NUMBER(SUBSTR(e.FC, 2))), 0) INTO maxE FROM
        EmployeeTB e;

    — 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;
    END LOOP;
END;
```

```

        END IF;
    END IF;

    cc := LPAD(TO_CHAR(maxE + i), 15, '0');
    INSERT INTO EmployeeTB VALUES (
        'E' || TO_CHAR(cc),
        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;
END;
```

```
    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					
1	Plan hash value: 1254032843				
2					
3	-----				
4	Id	Operation	Name	Rows	Bytes
5	-----				
6	0	INSERT STATEMENT		1	100
7	1	LOAD TABLE CONVENTIONAL	ORDERTB		
8	2	TABLE ACCESS BY INDEX ROWID	BUSINESSACCOUNTTB	1	29
9	* 3	INDEX UNIQUE SCAN	SYS_C0024322	1	
10	-----				
11					
12	Predicate Information (identified by operation id):				
13	-----				
14					
15	3	- access("B"."CODE"='B000000001')			

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
INSERT STATEMENT				1
LOAD TABLE CONVENTIONAL	ORDERTB			1
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					
1	Plan hash value: 3482816316				
2					
3	-----				
4	Id	Operation	Name	Rows	Bytes
5	-----				
6	0	UPDATE STATEMENT		1	34
7	1	UPDATE	ORDERTB		
8	* 2	INDEX UNIQUE SCAN	SYS_C0024335	1	34
9	3	TABLE ACCESS BY INDEX ROWID	TEAMTB	1	49
10	* 4	INDEX UNIQUE SCAN	SYS_C0024302	1	
11	-----				
12					
13	Predicate Information (identified by operation id):				
14	-----				
15					
16	2	- access("O"."ID"='000001111')			
17	4	- access("T"."ID"='T000001')			

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
UPDATE STATEMENT				1
UPDATE	ORDERTB			3
INDEX	SYS_C0024335	UNIQUE SCAN	1	1
Access Predicates				
O.ID='000001111'				
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					
1	Plan hash value: 2943281028				
2					
3					
4	Id	Operation	Name	Rows	Bytes
5					
6	0	SELECT STATEMENT		1	40
7	1	TABLE ACCESS BY INDEX ROWID	TEAMTB	1	40
8	* 2	INDEX UNIQUE SCAN	SYS_C0024302	1	
9					
10					
11	Predicate Information (identified by operation id):				
12					
13					
14	2	access("ID"='2C6535D4B8FF9D06E063020012AC4D2C')			

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				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					
1	Plan hash value: 2930154078				
2					
3					
4	Id	Operation	Name	Rows	Bytes
5					
6	0	SELECT STATEMENT		1	40
7	1	SORT AGGREGATE		1	40
8	* 2	TABLE ACCESS FULL	ORDERTB	1	40
9					
10					
11	Predicate Information (identified by operation id):				
12					
13					
14	2	filter("O"."TEAM"."ID"='2C6535D4B8FF9D06E063020012AC4D2C')			

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				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								
1	Plan hash value: 4254525359							
2								
3	-----							
4	Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	
5	-----							
6	0	SELECT STATEMENT		1	92	3 (34)	00:00:01	
7	1	SORT ORDER BY		1	92	3 (34)	00:00:01	
8	2	TABLE ACCESS FULL	TEAMTB	1	92	2 (0)	00:00:01	
9	-----							
10								
11	Note							
12	-----							
13	- 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 code centralizes database access in a separate `database.py` module. Each route follows the typical Flask pattern of "GET to render form, POST to handle form input."

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.

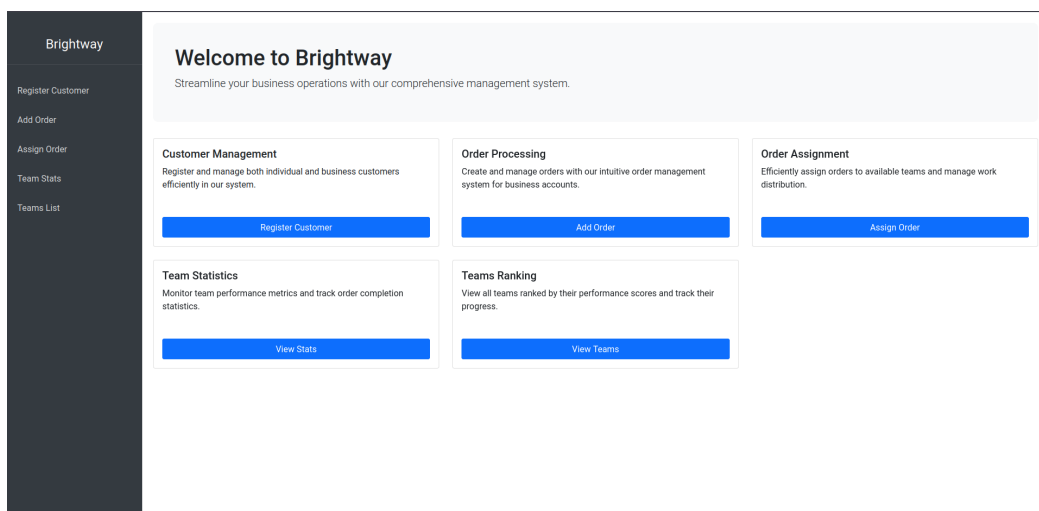


Figure 9.1: Home page of the web application.

The screenshot shows the 'Register Customer' form for individual customers. On the left is a dark sidebar with the 'Brightway' logo and a menu containing 'Register Customer', 'Add Order', 'Assign Order', 'Team Stats', and 'Teams List'. The main form area is titled 'Register Customer' and contains the following fields: 'VAT (11 characters)', 'Phone', and 'Email (max 50 characters)' at the top; 'Customer Type' with a dropdown menu set to 'Individual'; 'Name', 'Surname', and 'Date of Birth' (with a date picker showing 'gg / mm / aaaa') below. A blue 'Register Customer' button is at the bottom right.

Figure 9.2: Customer registration form for individual customers.

The screenshot shows the 'Register Customer' form for business customers. The sidebar is identical to Figure 9.2. The main form area is titled 'Register Customer' and contains the following fields: 'VAT (11 characters)', 'Phone', and 'Email (max 50 characters)' at the top; 'Customer Type' with a dropdown menu set to 'Business'; 'Company Name' below; and a section for address details including 'Street', 'Civic Number' (with a small icon), 'City', 'Province', 'Region', and 'State'. A blue 'Register Customer' button is at the bottom right.

Figure 9.3: Customer registration form for business customers.

The screenshot shows the 'Add New Order' form. The sidebar is identical to the previous figures. The main form area is titled 'Add New Order' and contains the following fields: 'Order ID (XXXXXXXXXX)' at the top; 'Placing Date' (with a date picker showing 'gg / mm / aaaa'); 'Order Mode' with a dropdown menu set to 'Online'; 'Order Type' with a dropdown menu set to 'Regular'; 'Cost'; and 'Business Account' with a dropdown menu set to 'Choose an account...'. A blue 'Add Order' button is at the bottom left.

Figure 9.4: Order insertion form.

The screenshot shows the 'Assign Order to Team' form. On the left is a dark sidebar with the 'Brightway' logo and a menu containing 'Register Customer', 'Add Order', 'Assign Order' (highlighted), 'Team Stats', and 'Teams List'. The main content area has the title 'Assign Order to Team'. Below the title is a form with two dropdown menus: 'Order ID' with the placeholder 'Choose an order...' and 'Team ID' with the placeholder 'Choose a team...'. A blue 'Assign Order' button is positioned below the 'Team ID' dropdown.

Figure 9.5: Order assignment form.

The screenshot shows the 'Team Statistics' page. The sidebar is identical to the previous figure. The main content area has the title 'Team Statistics'. Below the title is a 'Select Team' dropdown menu with 'T158562' selected. A blue 'Get Statistics' button is below the dropdown. Underneath, the text 'Statistics for Team T158562' is displayed. Below this, there are two rows of statistics: 'Total Orders Handled: 4' and 'Total Order Cost: €21762.55'.

Figure 9.6: Team statistics page.

The screenshot shows the 'Teams Ranking' page. The sidebar is identical to the previous figures. The main content area has the title 'Teams Ranking'. Below the title is a table with two columns: 'Team ID' and 'Performance Score'. The table contains 10 rows of data. At the bottom of the table are two blue buttons: 'Previous' on the left and 'Next' on the right.

Team ID	Performance Score
T729467	5.0
T204472	5.0
T187238	4.5
T529982	4.0
T748876	4.0
T487992	4.0
T513783	4.0
T321753	4.0
T187884	4.0
T308913	4.0

Figure 9.7: List of teams sorted by performance score.