Goodwill

TEAM NAME – FLASH DB

DATE – DEC 4, 2024

MEET THE TEAM

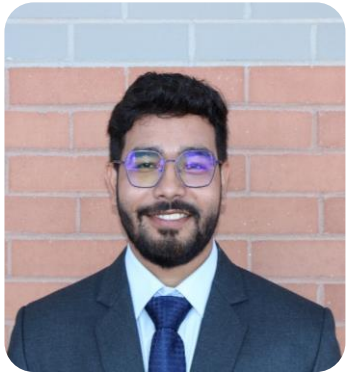
DEVANG

AISHWARYA

JANHAVI

ASHISH

OJASWITA



ROSHAN

AGENDA

- Client Requirements
- ER Diagram Construction
- ER to Relational and Normalization
- SQL Special Purpose Queries
- PL/SQL Triggers and Stored Procedures
- Frontend Demo

CLIENT REQUIREMENTS

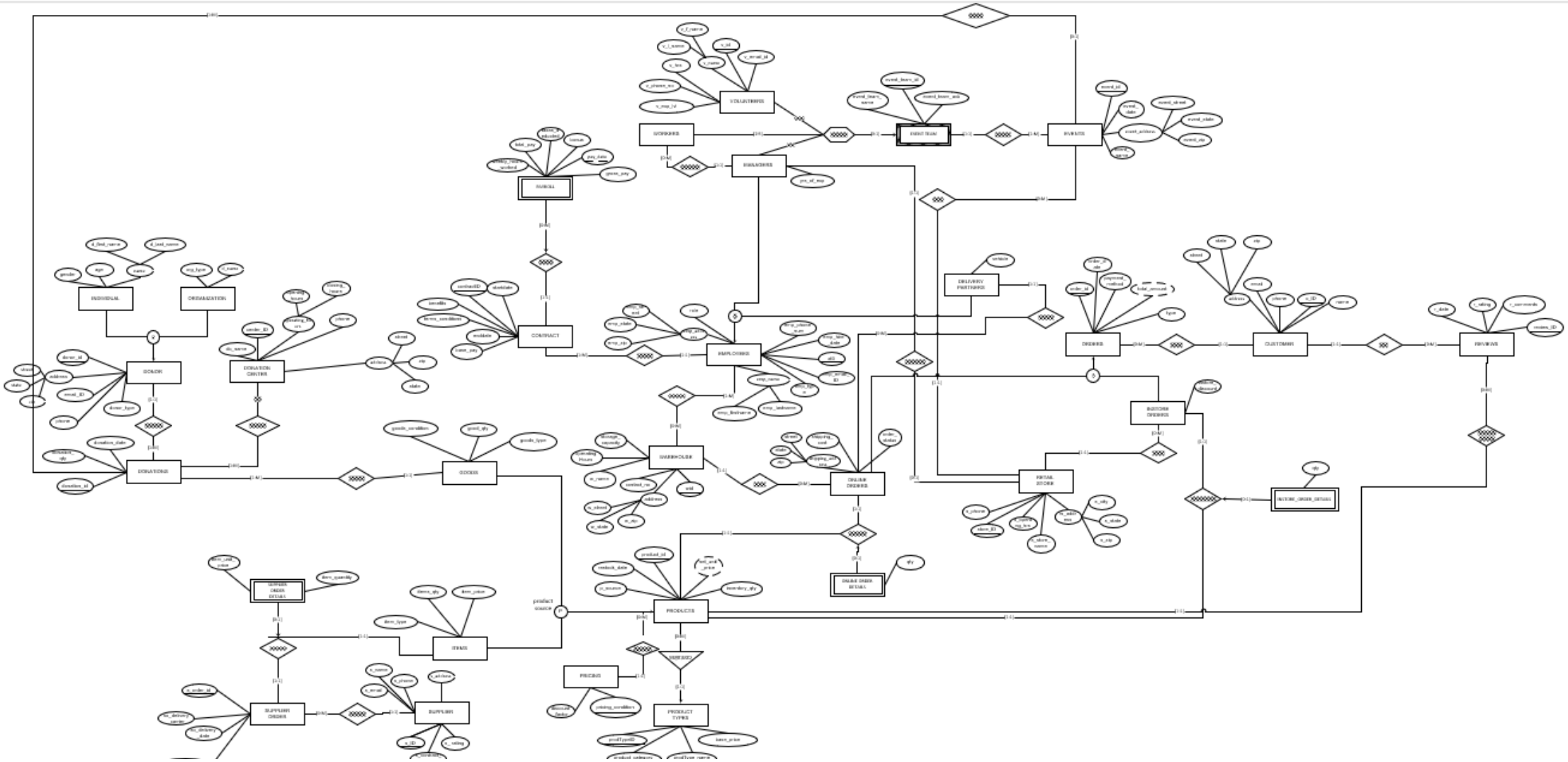
Employee Data Management

Tracking and Management of inventory, orders, and donations

Event Management

Data Analytics for Decision-Making

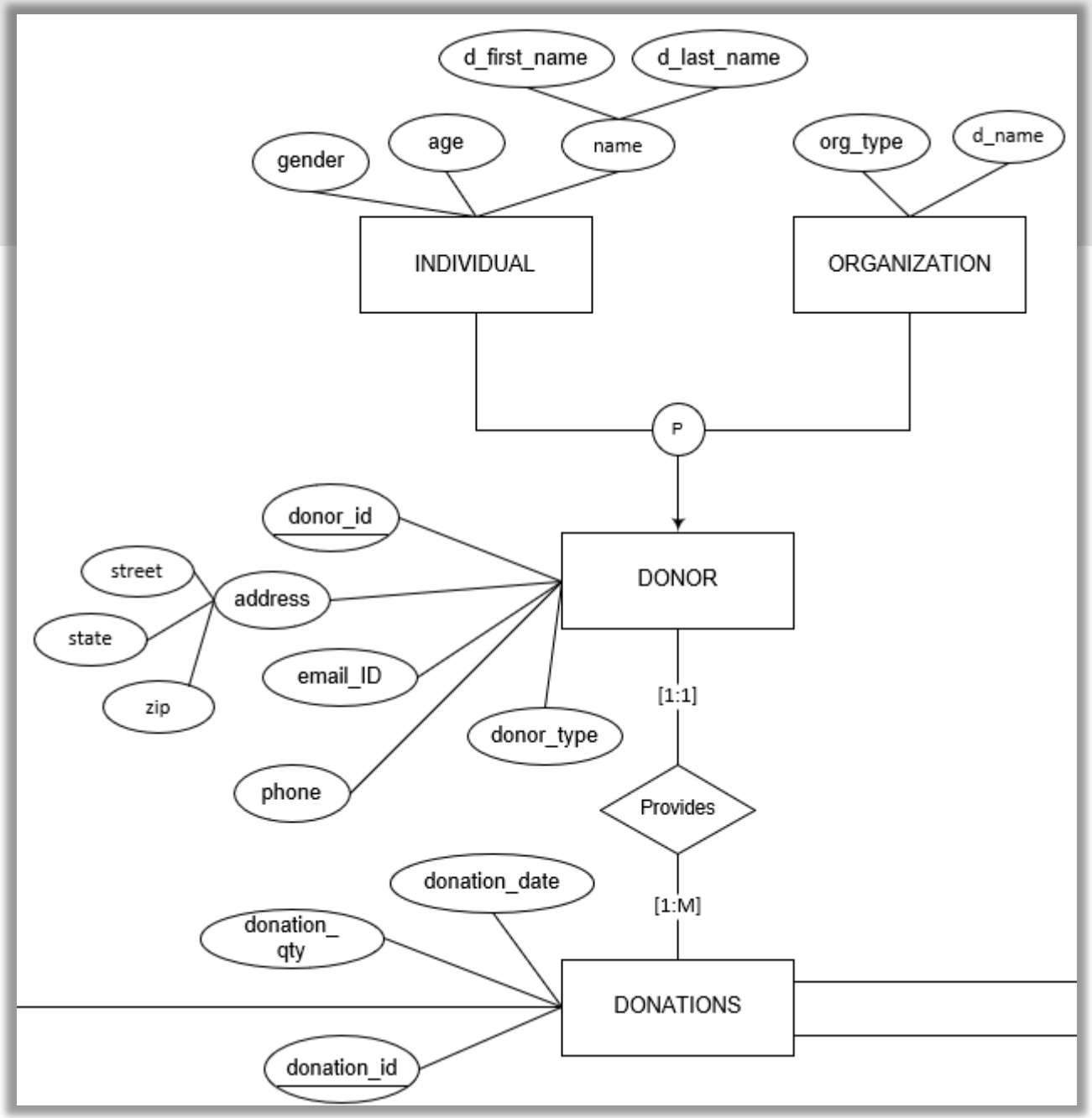
ER Diagram



ER Diagram

Donation System

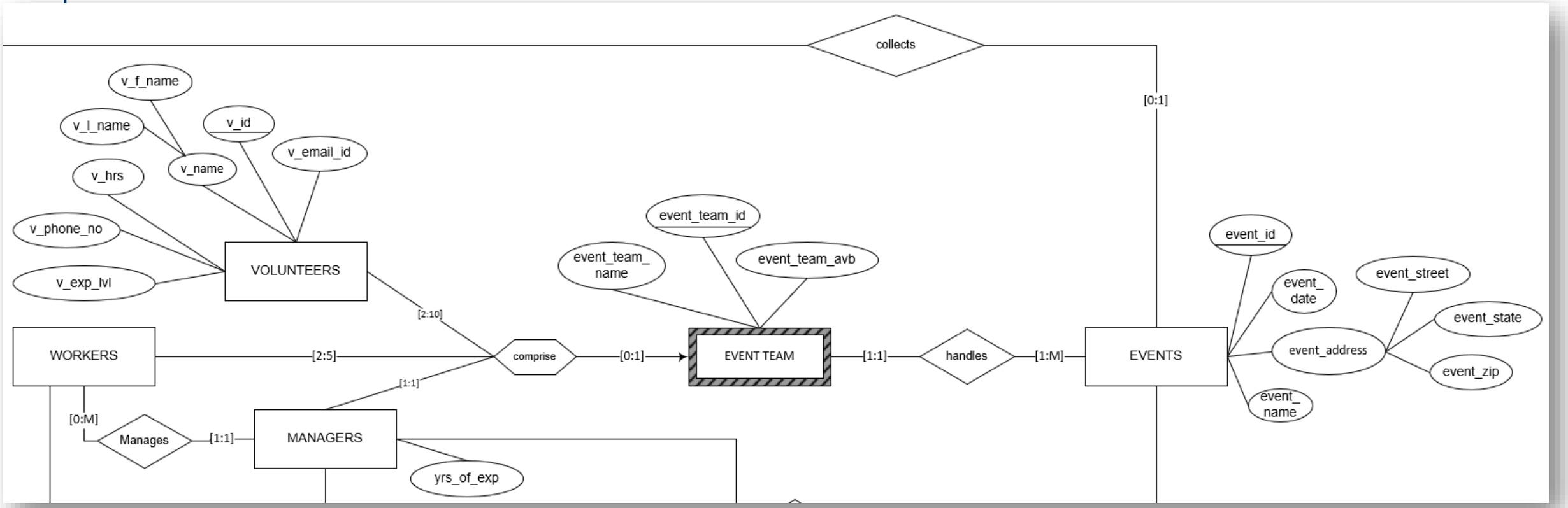
- Partition cardinality
- Composite attributes



ER Diagram

Events & Event Teams

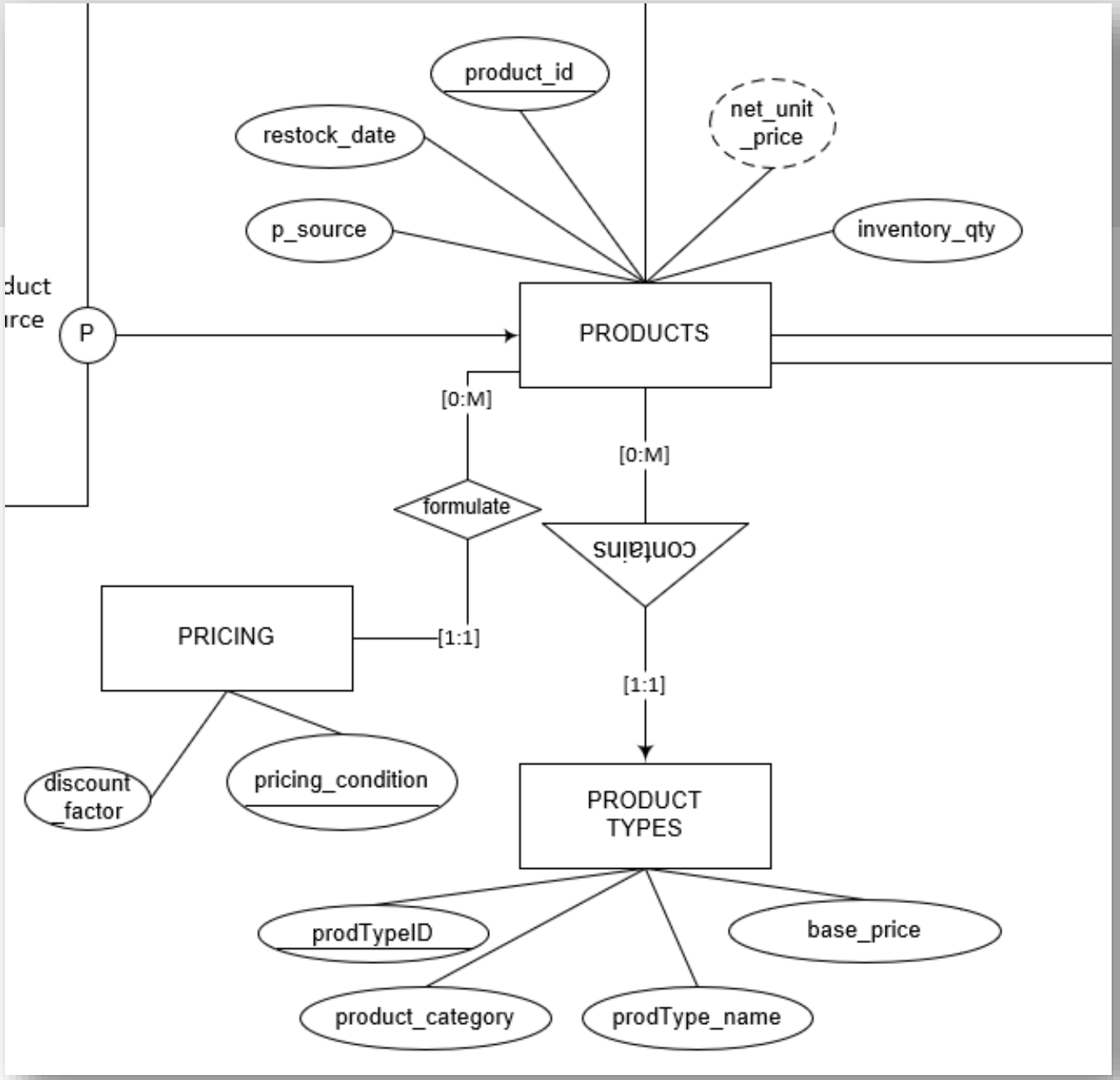
- Aggregation entity



ER Diagram

Products & Pricing System

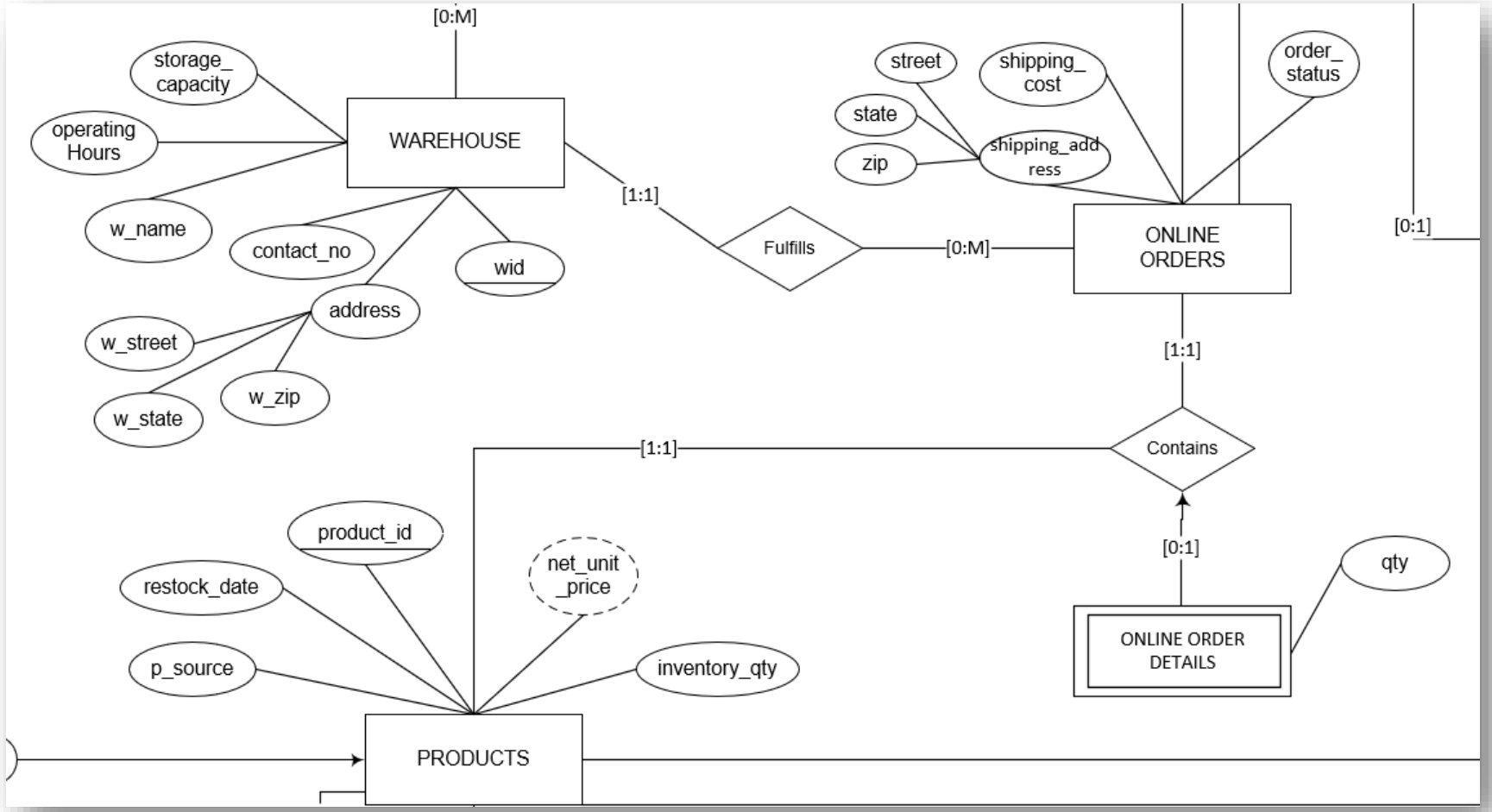
- Typing and Instantiation Classes
- Derived attributes



ER Diagram

ORDERS SYSTEMS

- Weak entity class
- Derived Attributes



ER to Relational

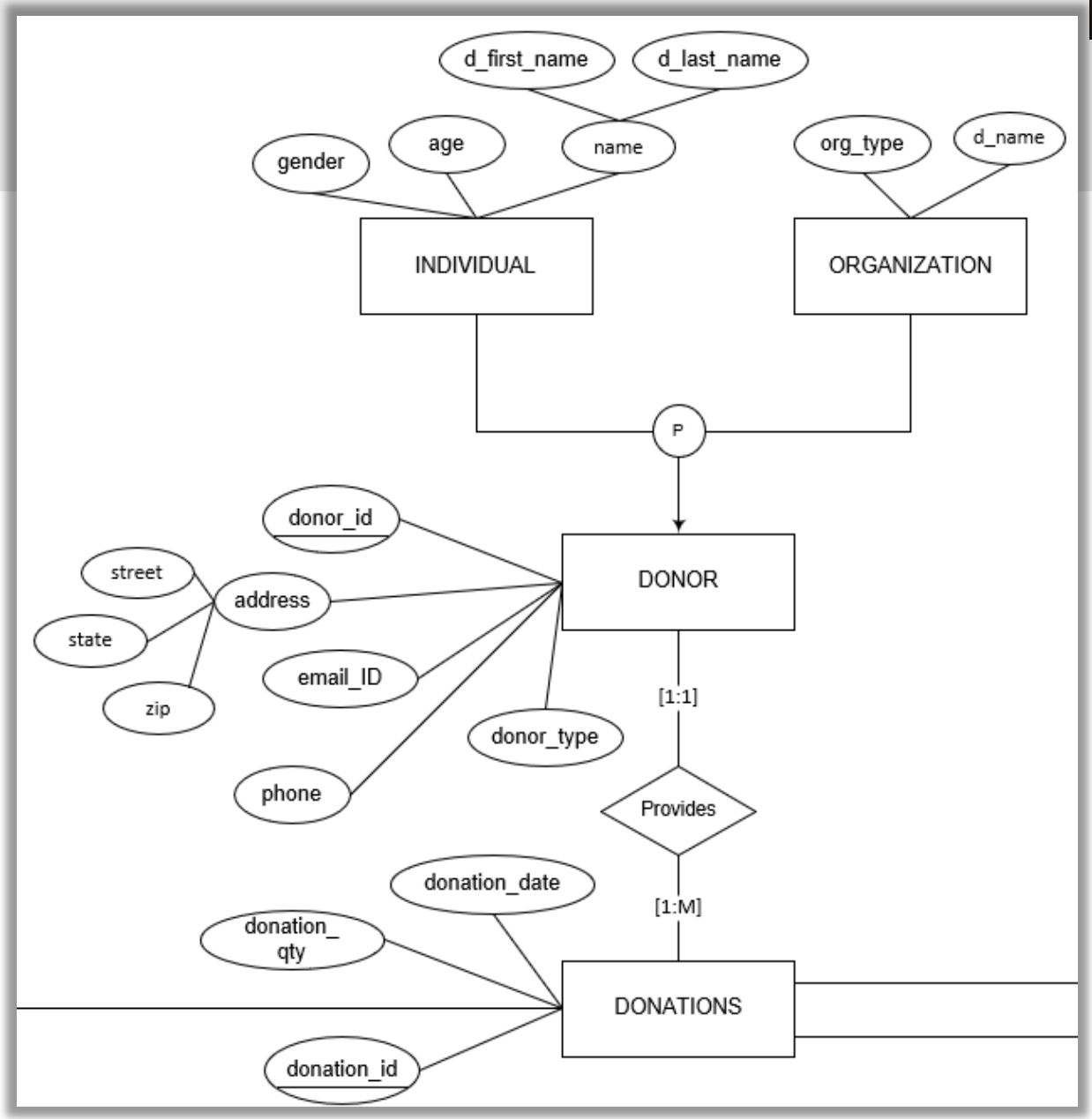
DONOR (donor_ID, email_ID, phone, street, state, zip, donor_type)

INDIVIDUAL_DONOR (donor_ID, d_first_name, d_last_name, age, gender)

Foreign key (donor_ID) references DONOR (donor_ID)

ORGANIZATION_DONOR (donor_ID, org_type, d_name)

Foreign key (donor_ID) references DONOR (donor_ID)



ER to Relational

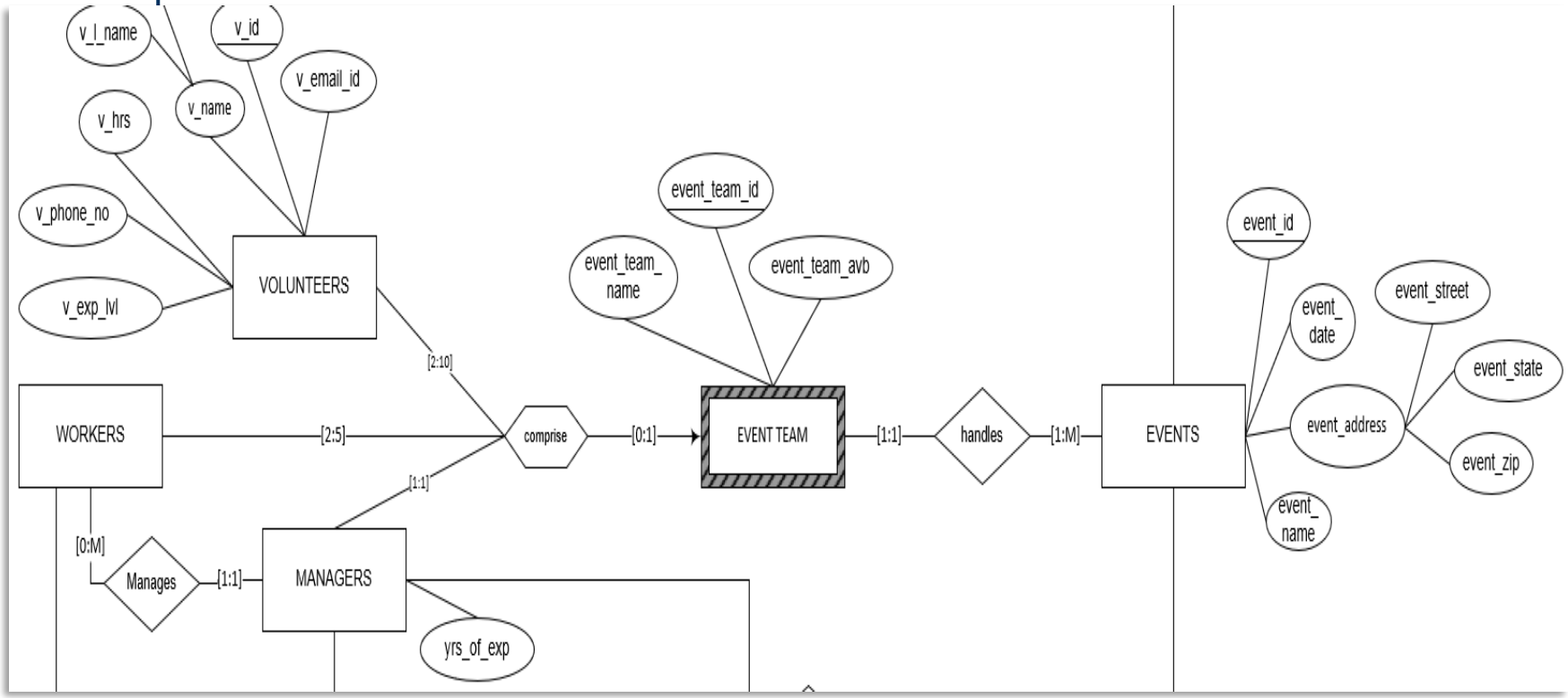
EVENT_TEAM (event_team_id, event_team_name, mgr_id, event_team_avb)

Foreign Key (mgr_id) REFERENCES MANAGERS (EID)

EVENTS (event_id, event_date, event_name, event_address, event_team_id, store_id)

Foreign Key (event_team_id) references EVENT_TEAMS (event_team_id)

Foreign Key (store_id) references RETAIL_STORES (store_id)



COMPRISE_WORKERS (event_team_id, eid)

Foreign Key (eid) references WORKERS (eid)

Foreign Key (event_team_id) references EVENT_TEAMS (event_team_id)

COMPRISE_VOLUNTEERS (event_team_id, v_id)

Foreign Key (v_id) references VOLUNTEERS (v_id)

Foreign Key (event_team_id) references EVENT_TEAMS (event_team_id)

Special Purpose SQL Queries

Top Donor along with their total quantity donated

```
WITH dc_qty AS (
  SELECT donor_ID, SUM(donation_qty) AS Sum_qty
  FROM donations
  GROUP BY donor_ID
)
SELECT d.d_first_name || ' ' || d.d_last_name AS Top_Donor, Sum_qty AS Qty_Donated
FROM individual_donor d
JOIN donations d1 ON d.donor_ID = d1.donor_ID
JOIN dc_qty d2 ON d2.donor_ID = d1.donor_ID
WHERE d2.Sum_qty = (SELECT MAX(Sum_qty) FROM dc_qty)
GROUP BY d.d_first_name, d.d_last_name, sum_qty;
```

Special Purpose SQL Queries

% Split of Donation w.r.t Source – Individuals or Organizations or Events

```
WITH allDonationsDetails AS (
    SELECT do.donor_id, donor_type, event_id, donation_qty, donation_id
    FROM DONATIONS do INNER JOIN DONOR don ON do.donor_id = don.donor_id
),
allDataCount(totalDonations) AS (
    SELECT SUM(donation_qty) FROM DONATIONS
),
seperateDonations AS (
SELECT
    SUM(CASE WHEN event_id IS NULL AND donor_type = 'INDV' THEN donation_qty END) AS Indv_donors,
    SUM(CASE WHEN event_id IS NULL AND donor_type = 'ORG' THEN donation_qty END) AS Org_donors,
    SUM(CASE WHEN event_id IS NOT NULL THEN donation_qty END) AS event_donors
FROM allDonationsDetails
)
SELECT ROUND(Indv_donors/totalDonations*100,2) "Individual Donations %",
ROUND(Org_donors/totalDonations*100,2) "Organization Donations %",
ROUND(event_donors/totalDonations*100,2) "Event Donations %"
FROM seperateDonations,allDataCount
;
```

Special Purpose SQL Queries

Best performing retail store and product type

```
SELECT
    r.s_store_name,
    p1.prodtype_name AS Product_Name,
    to_char(SUM(od.qty * p.net_unit_price), '$999999.99') AS total_revenue
FROM RETAIL_STORES r
LEFT JOIN INSTORE_ORDERS io ON r.store_id = io.store_id
LEFT JOIN INSTORE_ORDER_DETAILS od ON od.instore_order_id = io.instore_order_id
LEFT JOIN PRODUCTS p ON p.product_id = od.product_id
LEFT JOIN PRODUCT_TYPES p1 ON p1.prodtypeID = p.prodtypeID
GROUP BY CUBE(p1. prodtype_name, s_store_name)
ORDER BY total_revenue desc;
```

PL/SQL Triggers

Functionality:

- Retrieves the latest modified order detail item, calculates the price, and adds the value to the running total on the orders table.

```
CREATE OR REPLACE TRIGGER InstoreDetailsAFTERTrigger2
AFTER INSERT OR UPDATE OR DELETE ON INSTORE_ORDER_DETAILS
FOR EACH ROW
DECLARE
    v_additional_amount orders.total_amount%type;
    v_discount instore_orders.instore_discount%type;
    v_instore_order_id instore_order_details.instore_order_id%type;
    v_existing_amount orders.total_amount%type;
    v_new_total orders.total_amount%type;
    v_changedqty instore_order_details.qty%type;
    v_changedUnitPrice products.net_unit_price%type;
    v_productID instore_order_details.product_id%type;
BEGIN
    v_additional_amount := 0;

    IF (INSERTING) THEN
        v_changedqty := :new.qty;
        v_productID := :new.product_id;
        v_instore_order_id := :new.instore_order_id;
    ELSIF (UPDATING) THEN
        v_changedqty := :new.qty - :old.qty;
        v_productID := :new.product_id;
        v_instore_order_id := :new.instore_order_id;
    ELSE
        v_changedqty := :old.qty*(-1);
        v_productID := :old.product_id;
        v_instore_order_id := :old.instore_order_id;
    END IF;

    DBMS_OUTPUT.PUT_LINE ('v_additional_amount :' || v_additional_amount);

    SELECT net_unit_price INTO v_changedUnitPrice
    FROM PRODUCTS
    WHERE product_id = v_productID;

    v_additional_amount := v_changedqty*v_changedUnitPrice;

    DBMS_OUTPUT.PUT_LINE ('v_additional_amount 2: ' || v_additional_amount);

    SELECT total_amount INTO v_existing_amount
    FROM ORDERS
    WHERE order_id = v_instore_order_id;

    DBMS_OUTPUT.PUT_LINE ('v_existing_amount :' || v_existing_amount);

    v_discount := 0;
    IF (v_existing_amount IS NULL) THEN
        SELECT instore_discount
        INTO v_discount
        FROM INSTORE_ORDERS
        WHERE instore_order_id = v_instore_order_id;

        v_existing_amount := 0;
    END IF;

    DBMS_OUTPUT.PUT_LINE ('v_existing_amount 2 :' || v_existing_amount);
    DBMS_OUTPUT.PUT_LINE ('v_discount :' || v_discount);

    -- Subtract the discount from the total amount
    v_new_total := v_existing_amount + v_additional_amount - v_discount;

    DBMS_OUTPUT.PUT_LINE ('v_new_total :' || v_new_total);

    UPDATE ORDERS
    SET total_amount = v_new_total
    WHERE order_id = v_instore_order_id;
END;
/
```

PL/SQL Triggers

Functionality:

- Manages and newly added donations towards goods and products by handling inventory quantity.
- Utilizes ROW level trigger.

```
create or replace TRIGGER DonationAfterTrigger
AFTER INSERT ON DONATIONS
FOR EACH ROW
DECLARE
    v_productid donations.product_ID%type;
    v_productQty donations.donation_qty%type;
    v_originalGoodsQty goods.goods_qty%type;
    v_donationQty donations.donation_qty%type := :new.donation_qty;
BEGIN
    SELECT goods_qty, product_id INTO v_originalGoodsQty, v_productid
    FROM GOODS
    WHERE goods_type = :new.goods_type AND
    goods_condition = :new.goods_condition;

    UPDATE GOODS SET goods_qty = (v_originalGoodsQty + v_donationQty)
    WHERE product_id = v_productid;

    SELECT inventory_qty INTO v_productQty
    FROM PRODUCTS
    WHERE product_id = :new.product_id;

    UPDATE PRODUCTS SET inventory_qty = (v_productQty + v_donationQty)
    WHERE product_id = v_productid;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE (' Your SELECT statement retrieved no rows.
        Consider using a cursor.');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE (' Your SELECT statement retrieved multiple
        rows. Consider using a cursor.');
END;
```

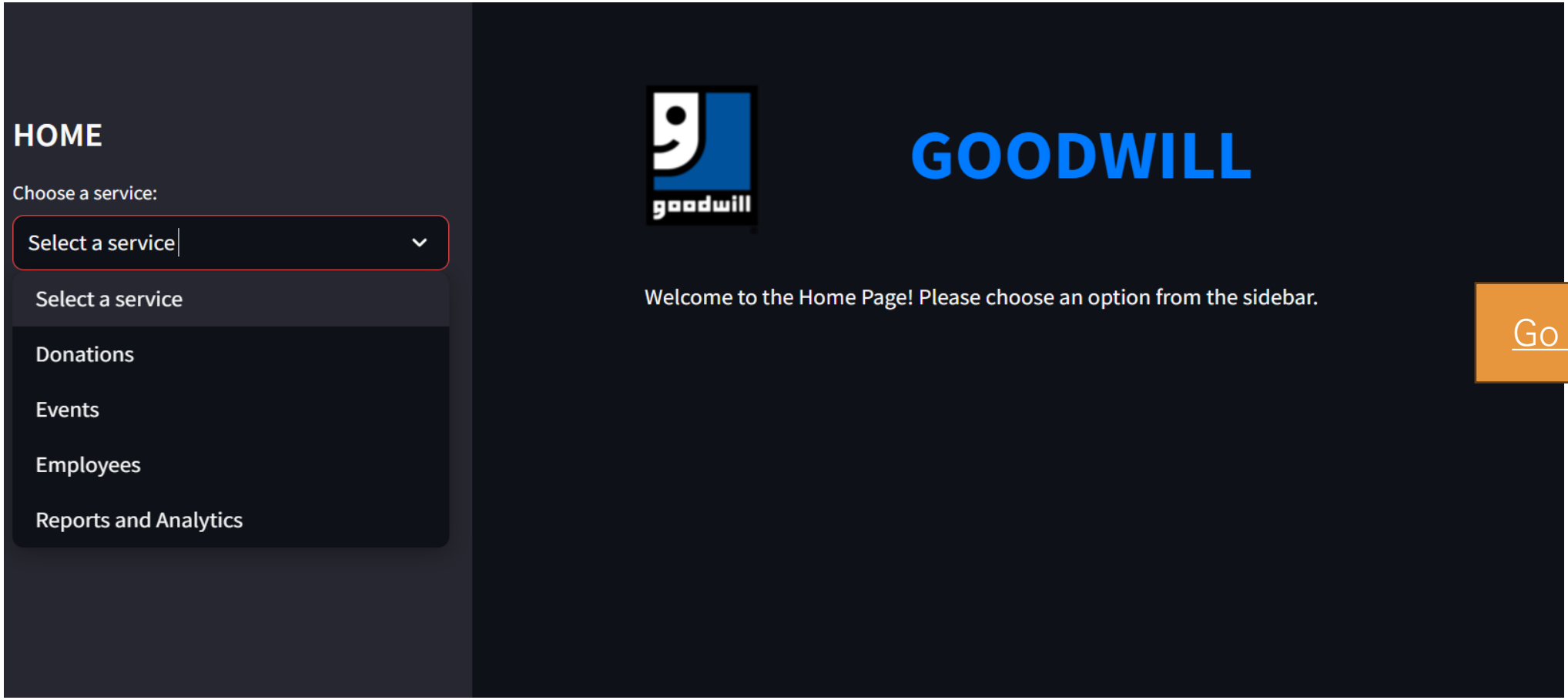
PL/SQL Procedures

Functionality:

- Validates Team availability to be assigned to events.

```
create or replace PROCEDURE AssignTeamToEvent (
    p_event_id IN VARCHAR2,
    p_team_id IN VARCHAR2
) AS
    v_availability NUMBER;
BEGIN
    -- Check to see if team is available
    SELECT EVENT_TEAM_AVAILABILITY
    INTO v_availability
    FROM EVENT_TEAMS
    WHERE EVENT_TEAM_ID = p_team_id;

    -- If the team is available, assign it to the event and update availability
    IF v_availability = 1 THEN
        -- Mark the team as unavailable
        UPDATE EVENT_TEAMS
        SET EVENT_TEAM_AVAILABILITY = 0
        WHERE EVENT_TEAM_ID = p_team_id;

        -- Link the team to the event
        UPDATE EVENTS
        SET EVENT_TEAM_ID = p_team_id
        WHERE EVENT_ID = p_event_id;

        DBMS_OUTPUT.PUT_LINE('Team ' || p_team_id || ' has been assigned to event ' ||
        p_event_id || ' and marked as unavailable.');
    ELSE
        -- If the team is already unavailable
        DBMS_OUTPUT.PUT_LINE('Team ' || p_team_id || ' is already unavailable.');
    END IF;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        -- Handle the case when the team ID does not exist
        DBMS_OUTPUT.PUT_LINE('Error: Team ID ' || p_team_id || ' does not exist in the
        EVENT_TEAMS table.');
    WHEN OTHERS THEN
        -- Handle other unexpected errors
        DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);
END;
```

Frontend – Admin Portal



Go to Demo

Q&A