CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

UNIVERSITY INSTITUTE OF COMPUTING

CASE STUDY REPORT
ON
Charity Donation Tracker

Program Name: BCA

Subject Name/Code: Database Management System (23CAT-251)

Submitted by:

Name: NAMAN SAINI

UID: 23BCA10698

Section: 4 - B

Submitted to:

Name: Arvinder Singh

Designation:

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

ABSTRACT

- **Introduction:**

Charity Donation Tracker :

The **Charity Donation Tracker** is a database management system designed to streamline the donation process for charitable organizations. In the modern world, where transparency and accountability are essential, this system helps manage donor records, track donations, run multiple campaigns, and monitor fund allocations to beneficiaries.

The system provides a structured way to store, access, and analyze data related to donations and their utilization. It eliminates manual record-keeping, reduces the chances of errors, and ensures real-time tracking of funds.

By implementing this system, NGOs and charitable institutions can ensure better planning of donation campaigns, efficient fund distribution, and improved trust among donors and beneficiaries. This project is a practical application of relational database concepts such as ER modeling, primary and foreign keys, normalization, and SQL operations.

Key features include:

- Multi-profile user accounts with personalized watchlists and viewing history
- Hierarchical content structure (TV shows → seasons → episodes)
- Content categorization through genre and people (actors, directors) relationships
- Recommendation engine with both content-based and user-based algorithms
- Subscription and payment processing with multiple plan options
- Device management and session tracking for security
- Performance optimization through indexes, stored procedures, and views

The database includes sample data, stored procedures for generating recommendations, triggers for rating updates, and views for trending content, making it production-ready for a streaming platform project.

- **Technique:**

The movie streaming platform database employs several key database design techniques:5.

Relational Database Design

The system uses a relational model with normalized tables representing core entities such as **Donors**, **Donations**, **Campaigns**, **Beneficiaries**, and **Allocations**. Primary and foreign key constraints are applied to maintain relational integrity across tables.

Database Normalization

All tables are designed to satisfy **Third Normal Form (3NF)**, minimizing redundancy

and ensuring that each attribute is functionally dependent on the primary key. This improves data consistency and simplifies updates.

 Many-to-Many Relationship Handling

The system handles complex relationships—such as a **donation supporting multiple beneficiaries**—through **junction tables** like Allocation, which connects Donation and Beneficiary.

 Performance Optimization

To enhance query efficiency, the database incorporates **indexes** on frequently accessed columns such as DonorID, DonationID, and CampaignID. Additionally, **views** can be created to simplify reporting, and **stored procedures** may be used for automated fund allocation or reporting logic.

 Data Integrity Enforcement

The system employs **constraints** (e.g., NOT NULL, CHECK, UNIQUE) and **foreign key rules** (ON DELETE CASCADE) to ensure data accuracy and prevent orphan records.

 Secure and Scalable Design

Designed to accommodate future expansion, the schema allows for new campaigns, recurring donations, and beneficiary tracking without structural changes. Access to sensitive data (like donor contact details) can be managed using view-based permissions.

System Configuration:

Based on the requirements of the **Charity Donation Tracker** database system, the following configuration is recommended to ensure performance, scalability, and data integrity:

1. **Database System:**
   MySQL or MariaDB is recommended due to its compatibility with structured relational data and support for constraints, triggers, stored procedures, and views.
2. **Storage Requirements:**
   A minimum of **5–10GB** storage is sufficient for initial records, with scalability up to **100GB+** for long-term donor, campaign, and beneficiary history tracking.

3. **Memory:**
   At least **8–16GB RAM** is advised to handle multi-table joins, aggregation queries, and real-time data reporting efficiently.

- **INPUT:**

```
-- Charity Donation Tracker Database
CREATE DATABASE Charity_Donation_Tracker;
USE Charity_Donation_Tracker;

-- Admins Table
CREATE TABLE Admins (
    AdminID SERIAL PRIMARY KEY,
    Username VARCHAR(50) UNIQUE NOT NULL,
    PasswordHash VARCHAR(255) NOT NULL,
    Email VARCHAR(100) UNIQUE NOT NULL,
    Role VARCHAR(50) CHECK (Role IN ('Manager', 'Clerk', 'Auditor')) NOT NULL,
    CreatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Donor_Types Table
CREATE TABLE Donor_Types (
    DonorTypeID SERIAL PRIMARY KEY,
    TypeName VARCHAR(50) NOT NULL UNIQUE
);

-- Cities Table
CREATE TABLE Cities (
    CityID SERIAL PRIMARY KEY,
    CityName VARCHAR(100) NOT NULL,
    State VARCHAR(100),
    Country VARCHAR(100)
);

-- Donors Table
CREATE TABLE Donors (
    DonorID SERIAL PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Email VARCHAR(100) NOT NULL UNIQUE,
    Phone VARCHAR(15),
    Address TEXT,
    CityID INT,
    DonorTypeID INT,
    FOREIGN KEY (CityID) REFERENCES Cities(CityID),
    FOREIGN KEY (DonorTypeID) REFERENCES Donor_Types(DonorTypeID)
);

-- Campaigns Table
CREATE TABLE Campaigns (
    CampaignID SERIAL PRIMARY KEY,
    Title VARCHAR(100) NOT NULL,
    Description TEXT,
    GoalAmount DECIMAL(10,2),
    StartDate DATE,
    EndDate DATE
```

```
);

-- Donations Table
CREATE TABLE Donations (
    DonationID SERIAL PRIMARY KEY,
    DonorID INT NOT NULL,
    CampaignID INT,
    Amount DECIMAL(10,2) NOT NULL,
    Date DATE DEFAULT CURRENT_DATE,
    PaymentMethod VARCHAR(50),
    FOREIGN KEY (DonorID) REFERENCES Donors(DonorID) ON DELETE CASCADE,
    FOREIGN KEY (CampaignID) REFERENCES Campaigns(CampaignID) ON DELETE SET NULL
);

-- Beneficiaries Table
CREATE TABLE Beneficiaries (
    BeneficiaryID SERIAL PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    NeedDescription TEXT,
    ContactInfo VARCHAR(100),
    Status VARCHAR(50) DEFAULT 'Pending'
);

-- Allocations Table
CREATE TABLE Allocations (
    AllocationID SERIAL PRIMARY KEY,
    DonationID INT NOT NULL,
    BeneficiaryID INT NOT NULL,
    AllocatedAmount DECIMAL(10,2) NOT NULL,
    Date DATE DEFAULT CURRENT_DATE,
    FOREIGN KEY (DonationID) REFERENCES Donations(DonationID) ON DELETE CASCADE,
    FOREIGN KEY (BeneficiaryID) REFERENCES Beneficiaries(BeneficiaryID) ON DELETE CASCADE
);

-- Donation Receipts Table
CREATE TABLE Donation_Receipts (
    ReceiptID SERIAL PRIMARY KEY,
    DonationID INT NOT NULL,
    ReceiptDate DATE DEFAULT CURRENT_DATE,
    FileURL VARCHAR(255),
    FOREIGN KEY (DonationID) REFERENCES Donations(DonationID)
);

-- Feedback Table
CREATE TABLE Feedback (
    FeedbackID SERIAL PRIMARY KEY,
    DonorID INT NOT NULL,
    Message TEXT NOT NULL,
    Rating INT CHECK (Rating >= 1 AND Rating <= 5),
    SubmittedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (DonorID) REFERENCES Donors(DonorID)
);

-- Payment Transactions Table
CREATE TABLE Payment_Transactions (
    TransactionID SERIAL PRIMARY KEY,
    DonationID INT NOT NULL,
```

```
    Amount DECIMAL(10,2) NOT NULL,
    PaymentStatus VARCHAR(20) CHECK (PaymentStatus IN ('Pending', 'Completed', 'Failed')),
    TransactionDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PaymentReference VARCHAR(100),
    FOREIGN KEY (DonationID) REFERENCES Donations(DonationID)
);

-- Login Activity Table
CREATE TABLE Login_Activity (
    LoginID SERIAL PRIMARY KEY,
    UserEmail VARCHAR(100),
    Role VARCHAR(20) CHECK (Role IN ('Donor', 'Admin')),
    LoginTime TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    IPAddress VARCHAR(45)
);
CREATE TABLE content_genres (
    content_id INTEGER NOT NULL REFERENCES content(content_id) ON DELETE CASCADE,
    genre_id INTEGER NOT NULL REFERENCES genres(genre_id) ON DELETE CASCADE,
    PRIMARY KEY (content_id, genre_id)
);

CREATE TABLE content_people (
    content_id INTEGER NOT NULL REFERENCES content(content_id) ON DELETE CASCADE,
    person_id INTEGER NOT NULL REFERENCES people(person_id) ON DELETE CASCADE,
    role VARCHAR(50) NOT NULL CHECK (role IN ('actor', 'director', 'producer', 'writer', 'cinematographer', 'composer', 'editor')),
    character_name VARCHAR(100), -- Only for actors
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (content_id, person_id, role)
);

CREATE TABLE viewing_history (
    history_id SERIAL PRIMARY KEY,
    profile_id INTEGER NOT NULL REFERENCES profiles(profile_id) ON DELETE CASCADE,
    content_id INTEGER REFERENCES content(content_id) ON DELETE SET NULL,
    episode_id INTEGER REFERENCES episodes(episode_id) ON DELETE SET NULL,
    start_time TIMESTAMP NOT NULL,
    end_time TIMESTAMP,
    watch_duration_seconds INTEGER,
    percentage_watched DECIMAL(5,2) CHECK (percentage_watched >= 0 AND percentage_watched <= 100),
    device_id VARCHAR(100),
    ip_address VARCHAR(45),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CHECK (content_id IS NOT NULL OR episode_id IS NOT NULL) -- Either content or episode must be specified
);
-- Table for Donor Categories
CREATE TABLE donor_categories (
    category_id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Table to Link Donors to Categories (Donor Segmentation)
CREATE TABLE donor_category_memberships (
    donor_id INTEGER NOT NULL REFERENCES donors(donor_id) ON DELETE CASCADE,
    category_id INTEGER NOT NULL REFERENCES donor_categories(category_id) ON DELETE CASCADE,
```

```
    joined_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (donor_id, category_id)
);

-- Table for Donation Distribution
CREATE TABLE donation_distribution (
    distribution_id SERIAL PRIMARY KEY,
    donation_id INTEGER NOT NULL REFERENCES donations(donation_id) ON DELETE CASCADE,
    charity_id INTEGER NOT NULL REFERENCES charities(charity_id) ON DELETE CASCADE,
    amount DECIMAL(10, 2) NOT NULL CHECK (amount > 0),
    distribution_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Table for Tracking Campaign Updates
CREATE TABLE campaign_updates (
    update_id SERIAL PRIMARY KEY,
    campaign_id INTEGER NOT NULL REFERENCES campaigns(campaign_id) ON DELETE CASCADE,
    update_title VARCHAR(100) NOT NULL,
    update_text TEXT,
    update_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Table for Thank You Notes Sent to Donors
CREATE TABLE thank_you_notes (
    note_id SERIAL PRIMARY KEY,
    donor_id INTEGER NOT NULL REFERENCES donors(donor_id) ON DELETE CASCADE,
    donation_id INTEGER NOT NULL REFERENCES donations(donation_id) ON DELETE CASCADE,
    note_text TEXT NOT NULL,
    note_sent_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    sent_by VARCHAR(100) -- Name or role of the person sending the note
);

-- Table for Donation Refunds
CREATE TABLE donation_refunds (
    refund_id SERIAL PRIMARY KEY,
    donation_id INTEGER NOT NULL REFERENCES donations(donation_id) ON DELETE CASCADE,
    refund_amount DECIMAL(10, 2) NOT NULL CHECK (refund_amount > 0),
    refund_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    refund_reason TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Table for Tracking Fundraising Events
CREATE TABLE fundraising_events (
    event_id SERIAL PRIMARY KEY,
    charity_id INTEGER NOT NULL REFERENCES charities(charity_id) ON DELETE CASCADE,
    event_name VARCHAR(100) NOT NULL,
    event_description TEXT,
    event_date TIMESTAMP NOT NULL,
    target_amount DECIMAL(10, 2) NOT NULL CHECK (target_amount >= 0),
    amount_raised DECIMAL(10, 2) DEFAULT 0,
    location VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
-- Table for Donor Testimonials
CREATE TABLE donor_testimonials (
    testimonial_id SERIAL PRIMARY KEY,
    donor_id INTEGER NOT NULL REFERENCES donors(donor_id) ON DELETE CASCADE,
    campaign_id INTEGER NOT NULL REFERENCES campaigns(campaign_id) ON DELETE CASCADE,
    testimonial_text TEXT NOT NULL,
    testimonial_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    is_approved BOOLEAN DEFAULT FALSE
);

-- Table for Charity Beneficiaries (People/Groups receiving donations)
CREATE TABLE beneficiaries (
    beneficiary_id SERIAL PRIMARY KEY,
    charity_id INTEGER NOT NULL REFERENCES charities(charity_id) ON DELETE CASCADE,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Table for Tracking Fund Allocation to Beneficiaries
CREATE TABLE fund_allocation (
    allocation_id SERIAL PRIMARY KEY,
    beneficiary_id INTEGER NOT NULL REFERENCES beneficiaries(beneficiary_id) ON DELETE CASCADE,
    campaign_id INTEGER NOT NULL REFERENCES campaigns(campaign_id) ON DELETE CASCADE,
    allocated_amount DECIMAL(10, 2) NOT NULL CHECK (allocated_amount > 0),
    allocation_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Table for Reporting - Donations Summary Report
CREATE TABLE donation_reports (
    report_id SERIAL PRIMARY KEY,
    report_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    total_donations DECIMAL(10, 2) NOT NULL,
    total_donors INTEGER NOT NULL,
    total_campaigns INTEGER NOT NULL,
    total_fund_allocated DECIMAL(10, 2),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Table for Tracking Gift Donations (Items given as donations, e.g., clothes, food)
CREATE TABLE gift_donations (
    gift_id SERIAL PRIMARY KEY,
    donor_id INTEGER NOT NULL REFERENCES donors(donor_id) ON DELETE CASCADE,
    charity_id INTEGER NOT NULL REFERENCES charities(charity_id) ON DELETE CASCADE,
    gift_description TEXT NOT NULL,
    quantity INTEGER NOT NULL CHECK (quantity > 0),
    gift_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    received_by VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Table for Recurring Donations
CREATE TABLE recurring_donations (
    recurring_donation_id SERIAL PRIMARY KEY,
    donor_id INTEGER NOT NULL REFERENCES donors(donor_id) ON DELETE CASCADE,
    charity_id INTEGER NOT NULL REFERENCES charities(charity_id) ON DELETE CASCADE,
```

```
    amount DECIMAL(10, 2) NOT NULL CHECK (amount > 0),
    donation_frequency VARCHAR(20) CHECK (donation_frequency IN ('weekly', 'monthly', 'quarterly', 'yearly')),
    start_date DATE NOT NULL,
    end_date DATE,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Table for Tax Deductible Donations
CREATE TABLE tax_deductible_donations (
    tax_donation_id SERIAL PRIMARY KEY,
    donation_id INTEGER NOT NULL REFERENCES donations(donation_id) ON DELETE CASCADE,
    tax_deduction_percentage DECIMAL(5, 2) NOT NULL CHECK (tax_deduction_percentage >= 0 AND tax_deduction_percentage <= 100),
    eligible_for_tax_deduction BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Table for Social Media Donations Tracking (When donors share campaigns on social media)
CREATE TABLE social_media_shares (
    share_id SERIAL PRIMARY KEY,
    donor_id INTEGER NOT NULL REFERENCES donors(donor_id) ON DELETE CASCADE,
    campaign_id INTEGER NOT NULL REFERENCES campaigns(campaign_id) ON DELETE CASCADE,
    platform VARCHAR(50) CHECK (platform IN ('facebook', 'twitter', 'instagram', 'linkedin', 'other')),
    share_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Index on donor name for quick search
CREATE INDEX idx_donors_name ON donors(name);

-- Index on donor email for quick search and lookup
CREATE INDEX idx_donors_email ON donors(email);

-- Index on charity name for quick search
CREATE INDEX idx_charities_name ON charities(name);

-- Index on campaign target amount for filtering campaigns by target amount
CREATE INDEX idx_campaigns_target_amount ON campaigns(target_amount);

-- Index on donation date for filtering donations by date
CREATE INDEX idx_donations_donation_date ON donations(donation_date);

-- Index on donation amount for filtering donations by amount
CREATE INDEX idx_donations_amount ON donations(amount);

-- Index on donor ID in donations table to quickly find donations by donor
CREATE INDEX idx_donations_donor_id ON donations(donor_id);

-- Index on campaign ID in donations table to quickly find donations by campaign
CREATE INDEX idx_donations_campaign_id ON donations(campaign_id);

-- Index on campaign ID in donation distribution to optimize query performance
CREATE INDEX idx_donation_distribution_campaign_id ON donation_distribution(campaign_id);

-- Index on charity ID in donation distribution to optimize query performance
CREATE INDEX idx_donation_distribution_charity_id ON donation_distribution(charity_id);
```

```
-- Index on donation receipt number for quick lookup
CREATE INDEX idx_donation_receipts_receipt_number ON donation_receipts(receipt_number);

-- Index on donor ID for tracking communication with donors
CREATE INDEX idx_donor_communications_donor_id ON donor_communications(donor_id);

-- Index on campaign ID for tracking updates related to specific campaigns
CREATE INDEX idx_campaign_updates_campaign_id ON campaign_updates(campaign_id);

-- Index on donation refund date to track refund history
CREATE INDEX idx_donation_refunds_refund_date ON donation_refunds(refund_date);

-- Index on fundraising event date for reporting and querying by event date
CREATE INDEX idx_fundraising_events_event_date ON fundraising_events(event_date);

-- Index on donor ID for collecting donor testimonials for quick lookup
CREATE INDEX idx_donor_testimonials_donor_id ON donor_testimonials(donor_id);

-- Index on campaign ID in donor testimonials to quickly filter testimonials by campaign
CREATE INDEX idx_donor_testimonials_campaign_id ON donor_testimonials(campaign_id);

-- Index on beneficiary ID to track fund allocation efficiently
CREATE INDEX idx_fund_allocation_beneficiary_id ON fund_allocation(beneficiary_id);

-- Index on campaign ID in fund allocation to track how funds are distributed in each campaign
CREATE INDEX idx_fund_allocation_campaign_id ON fund_allocation(campaign_id);

-- Index on donation report date to filter reports by date
CREATE INDEX idx_donation_reports_report_date ON donation_reports(report_date);

-- Index on gift donation donor ID for tracking donations of physical goods by donor
CREATE INDEX idx_gift_donations_donor_id ON gift_donations(donor_id);

-- Index on gift donation charity ID for tracking donations of physical goods by charity
CREATE INDEX idx_gift_donations_charity_id ON gift_donations(charity_id);

-- Index on recurring donation donor ID for filtering recurring donations by donor
CREATE INDEX idx_recurring_donations_donor_id ON recurring_donations(donor_id);

-- Index on recurring donation charity ID for filtering recurring donations by charity
CREATE INDEX idx_recurring_donations_charity_id ON recurring_donations(charity_id);

-- Index on tax-deductible donations donation ID for quick tax-related queries
CREATE INDEX idx_tax_deductible_donations_donation_id ON tax_deductible_donations(donation_id);

-- Index on social media shares donor ID for tracking shared campaigns
CREATE INDEX idx_social_media_shares_donor_id ON social_media_shares(donor_id);
-- Insert random data into donors table
INSERT INTO donors (name, email, phone, address, created_at)
VALUES
('John Doe', 'john.doe@example.com', '1234567890', '123 Elm Street', CURRENT_TIMESTAMP),
('Jane Smith', 'jane.smith@example.com', '9876543210', '456 Oak Avenue', CURRENT_TIMESTAMP),
('Emily Johnson', 'emily.johnson@example.com', '5551234567', '789 Pine Road', CURRENT_TIMESTAMP);

-- Insert random data into charities table
INSERT INTO charities (name, description, created_at)
VALUES
```

```
('Red Cross', 'A charity dedicated to disaster relief and humanitarian aid.', CURRENT_TIMESTAMP),
('Save the Children', 'Helping children worldwide with food, education, and healthcare.', CURRENT_TIMESTAMP),
('World Wildlife Fund', 'Protecting the planet's wildlife and environment.', CURRENT_TIMESTAMP);

-- Insert random data into campaigns table
INSERT INTO campaigns (charity_id, campaign_name, campaign_description, target_amount, start_date, end_date, created_at)
VALUES
(1, 'Disaster Relief Fund', 'Providing aid to victims of natural disasters', 50000.00, '2025-01-01', '2025-03-01', CURRENT_TIMESTAMP),
(2, 'Children Education Fund', 'Helping underprivileged children get an education', 30000.00, '2025-02-01', '2025-04-01',
CURRENT_TIMESTAMP),
(3, 'Wildlife Protection Fund', 'Protecting endangered species and habitats', 45000.00, '2025-03-01', '2025-06-01', CURRENT_TIMESTAMP);

-- Insert random data into donations table
INSERT INTO donations (donor_id, campaign_id, donation_amount, donation_date)
VALUES
(1, 1, 500.00, '2025-01-15'),
(2, 2, 200.00, '2025-02-10'),
(3, 3, 1000.00, '2025-03-05');

-- Insert random data into donation_distribution table
INSERT INTO donation_distribution (donation_id, charity_id, amount, distribution_date)
VALUES
(1, 1, 500.00, '2025-01-16'),
(2, 2, 200.00, '2025-02-11'),
(3, 3, 1000.00, '2025-03-06');

-- Insert random data into donation_receipts table
INSERT INTO donation_receipts (donation_id, receipt_number, receipt_date)
VALUES
(1, 'RCPT001', '2025-01-15'),
(2, 'RCPT002', '2025-02-10'),
(3, 'RCPT003', '2025-03-05');

-- Insert random data into donor_communications table
INSERT INTO donor_communications (donor_id, message, communication_date)
VALUES
(1, 'Thank you for your generous donation to the Disaster Relief Fund.', '2025-01-16'),
(2, 'Thank you for your contribution to the Children Education Fund.', '2025-02-11'),
(3, 'Your donation to the Wildlife Protection Fund is greatly appreciated.', '2025-03-06');

-- Insert random data into campaign_updates table
INSERT INTO campaign_updates (campaign_id, update_title, update_text, update_date)
VALUES
(1, 'Funds Raised', 'We have raised $10,000 towards our goal of $50,000 for disaster relief.', '2025-01-20'),
(2, 'Milestone Achieved', 'We've reached 50% of our target for children's education.', '2025-02-15'),
(3, 'Support Grows', 'The Wildlife Protection Fund has exceeded $30,000 in donations.', '2025-03-10');

-- Insert random data into donation_refunds table
INSERT INTO donation_refunds (donation_id, refund_amount, refund_date, refund_reason)
VALUES
(1, 50.00, '2025-01-20', 'Duplicate donation');

-- Insert random data into fundraising_events table
INSERT INTO fundraising_events (charity_id, event_name, event_description, event_date, target_amount, amount_raised, location)
VALUES
(1, 'Disaster Relief Gala', 'A gala to raise funds for disaster relief efforts.', '2025-01-25', 100000.00, 25000.00, 'New York City'),
```

```
(2, 'Children's Education Auction', 'An auction to raise funds for educational programs for children.', '2025-02-20', 50000.00, 15000.00, 'Los Angeles'),
(3, 'Wildlife Fundraiser', 'Fundraising event to protect wildlife and raise awareness.', '2025-03-15', 70000.00, 20000.00, 'Chicago');

-- Insert random data into donor_testimonials table
INSERT INTO donor_testimonials (donor_id, campaign_id, testimonial_text)
VALUES
(1, 1, 'I am proud to support the Disaster Relief Fund, especially during such difficult times.'),
(2, 2, 'Supporting education for children in need is something that is very important to me.'),
(3, 3, 'I believe in protecting wildlife, and I'm happy to support this cause.');

-- Insert random data into beneficiaries table
INSERT INTO beneficiaries (charity_id, name, description)
VALUES
(1, 'Hurricane Victims', 'Individuals and families affected by recent hurricanes.', CURRENT_TIMESTAMP),
(2, 'Underprivileged Children', 'Children in need of education, healthcare, and basic support.', CURRENT_TIMESTAMP),
(3, 'Endangered Species', 'Protection efforts for endangered animals and their habitats.', CURRENT_TIMESTAMP);

-- Insert random data into fund_allocation table
INSERT INTO fund_allocation (beneficiary_id, campaign_id, allocated_amount)
VALUES
(1, 1, 5000.00),
(2, 2, 4000.00),
(3, 3, 7000.00);

-- Insert random data into donation_reports table
INSERT INTO donation_reports (total_donations, total_donors, total_campaigns, total_fund_allocated)
VALUES
(20000.00, 50, 10, 15000.00),
(15000.00, 30, 8, 12000.00),
(30000.00, 70, 15, 25000.00);

-- Insert random data into gift_donations table
INSERT INTO gift_donations (donor_id, charity_id, gift_description, quantity)
VALUES
(1, 1, 'Clothing for disaster victims', 100),
(2, 2, 'Books and educational materials', 200),
(3, 3, 'Toys for children in orphanages', 50);

-- Insert random data into recurring_donations table
INSERT INTO recurring_donations (donor_id, charity_id, amount, donation_frequency, start_date, end_date)
VALUES
(1, 1, 100.00, 'monthly', '2025-01-01', NULL),
(2, 2, 50.00, 'monthly', '2025-02-01', NULL),
(3, 3, 200.00, 'quarterly', '2025-03-01', NULL);

-- Insert random data into tax_deductible_donations table
INSERT INTO tax_deductible_donations (donation_id, tax_deduction_percentage, eligible_for_tax_deduction)
VALUES
(1, 10.00, TRUE),
(2, 5.00, TRUE),
(3, 20.00, TRUE);

-- Insert random data into social_media_shares table
INSERT INTO social_media_shares (donor_id, campaign_id, platform)
VALUES
(1, 1, 'facebook'),
```

```
        (2, 2, 'twitter'),
        (3, 3, 'instagram');

DELIMITER //
CREATE PROCEDURE update_content_recommendations()
BEGIN
    DELETE FROM content_recommendations;

    INSERT INTO content_recommendations (source_content_id, recommended_content_id, similarity_score, recommendation_reason)
    SELECT
        c1.content_id AS source_content_id,
        c2.content_id AS recommended_content_id,
        COUNT(DISTINCT cg1.genre_id) /
            (SELECT COUNT(DISTINCT genre_id) FROM content_genres WHERE content_id = c1.content_id) AS similarity_score,
        CONCAT('Because you watched ', c1.title) AS recommendation_reason
    FROM
        content c1
    JOIN
        content_genres cg1 ON c1.content_id = cg1.content_id
    JOIN
        content_genres cg2 ON cg1.genre_id = cg2.genre_id
    JOIN
        content c2 ON cg2.content_id = c2.content_id
    WHERE
        c1.content_id != c2.content_id
    GROUP BY
        c1.content_id, c2.content_id, c1.title
    HAVING
        COUNT(DISTINCT cg1.genre_id) >= 2
    ORDER BY
        similarity_score DESC
    LIMIT 500;

    SELECT CONCAT('Content recommendations updated at ', NOW()) AS log_message;
END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE generate_user_recommendations(IN p_profile_id INTEGER)
BEGIN
    DECLARE v_genre_count INT;

    DROP TEMPORARY TABLE IF EXISTS temp_genre_preferences;
    CREATE TEMPORARY TABLE temp_genre_preferences (
        genre_id INT
    );

    DELETE FROM user_recommendations WHERE profile_id = p_profile_id;

    INSERT INTO temp_genre_preferences
    SELECT DISTINCT cg.genre_id
    FROM viewing_history vh
    JOIN content c ON vh.content_id = c.content_id
    JOIN content_genres cg ON c.content_id = cg.content_id
    WHERE vh.profile_id = p_profile_id;

    SELECT COUNT(*) INTO v_genre_count FROM temp_genre_preferences;
```

```
    IF v_genre_count = 0 THEN
        INSERT INTO temp_genre_preferences
        SELECT DISTINCT cg.genre_id
        FROM watchlist_items wi
        JOIN content_genres cg ON wi.content_id = cg.content_id
        WHERE wi.profile_id = p_profile_id;

        SELECT COUNT(*) INTO v_genre_count FROM temp_genre_preferences;
    END IF;

    IF v_genre_count = 0 THEN
        INSERT INTO user_recommendations (profile_id, content_id, score, recommendation_reason)
        SELECT
            p_profile_id,
            content_id,
            average_rating / 10 AS score,
            'Popular on StreamFlix'
        FROM
            content
        ORDER BY
            average_rating DESC, total_ratings DESC
        LIMIT 10;
    ELSE
        INSERT INTO user_recommendations (profile_id, content_id, score, recommendation_reason)
        SELECT
            p_profile_id,
            c.content_id,
            (COUNT(DISTINCT cg.genre_id) / v_genre_count * 0.7 +
             c.average_rating / 10 * 0.3) AS score,
            CASE
                WHEN COUNT(DISTINCT cg.genre_id) > v_genre_count / 2 THEN CONCAT('Because you like ', g.name)
                ELSE 'Recommended for you'
            END AS recommendation_reason
        FROM
            content c
        JOIN
            content_genres cg ON c.content_id = cg.content_id
        JOIN
            genres g ON cg.genre_id = g.genre_id
        WHERE
            cg.genre_id IN (SELECT genre_id FROM temp_genre_preferences)
            AND c.content_id NOT IN (
                SELECT content_id FROM viewing_history WHERE profile_id = p_profile_id AND content_id IS NOT NULL
            )
        GROUP BY
            c.content_id, c.average_rating, g.name
        ORDER BY
            score DESC
        LIMIT 15;
    END IF;

    SELECT CONCAT('User recommendations updated for profile ', p_profile_id, ' at ', NOW()) AS log_message;

    DROP TEMPORARY TABLE IF EXISTS temp_genre_preferences;
END //
DELIMITER ;
```

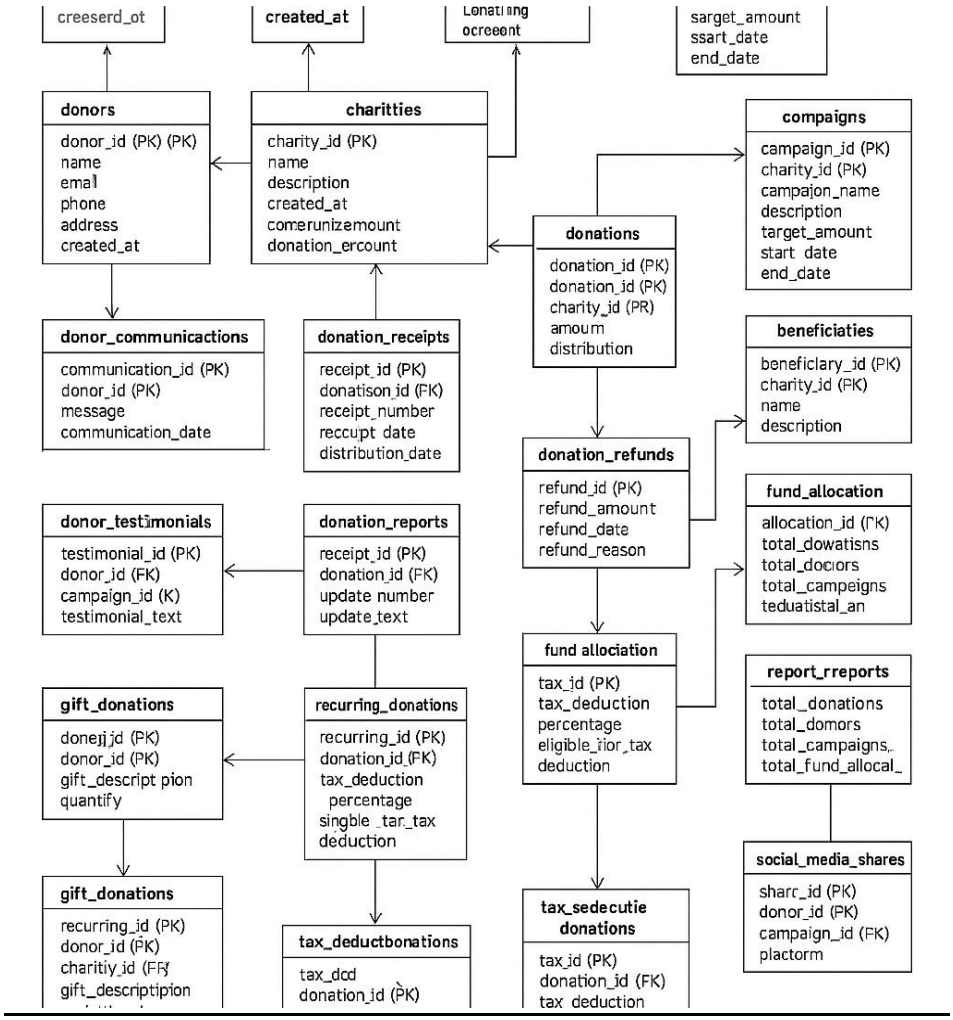
```
DELIMITER //
CREATE TRIGGER after_review_insert_or_update
AFTER INSERT ON content_reviews
FOR EACH ROW
BEGIN
   UPDATE content
   SET
      average_rating = (SELECT AVG(rating) FROM content_reviews WHERE content_id = NEW.content_id),
      total_ratings = (SELECT COUNT(*) FROM content_reviews WHERE content_id = NEW.content_id)
   WHERE content_id = NEW.content_id;
END //

CREATE TRIGGER after_review_update
AFTER UPDATE ON content_reviews
FOR EACH ROW
BEGIN
   UPDATE content
   SET
      average_rating = (SELECT AVG(rating) FROM content_reviews WHERE content_id = NEW.content_id),
      total_ratings = (SELECT COUNT(*) FROM content_reviews WHERE content_id = NEW.content_id)
   WHERE content_id = NEW.content_id;
END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE get_personalized_content(IN p_profile_id INTEGER, IN p_limit INTEGER)
BEGIN
   IF p_limit IS NULL THEN
      SET p_limit = 20;
   END IF;

   CALL generate_user_recommendations(p_profile_id);

   SELECT
      c.content_id,
      c.title,
      c.type,
      c.poster_url,
      ur.recommendation_reason,
      ur.score
   FROM
      user_recommendations ur
   JOIN
      content c ON ur.content_id = c.content_id
   WHERE
      ur.profile_id = p_profile_id
   ORDER BY
      ur.score DESC
   LIMIT p_limit;
END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE get_continue_watching(IN p_profile_id INTEGER, IN p_limit INTEGER)
BEGIN
   IF p_limit IS NULL THEN
```

```
    SET p_limit = 10;
  END IF;

  SELECT
    vh.content_id,
    vh.episode_id,
    COALESCE(c.title, (SELECT c2.title FROM episodes e JOIN seasons s ON e.season_id = s.season_id JOIN content c2 ON s.content_id =
c2.content_id WHERE e.episode_id = vh.episode_id)),
    COALESCE(c.poster_url, (SELECT c2.poster_url FROM episodes e JOIN seasons s ON e.season_id = s.season_id JOIN content c2 ON s.content_id
= c2.content_id WHERE e.episode_id = vh.episode_id)),
    vh.percentage_watched,
    vh.start_time AS last_watched
  FROM
    viewing_history vh
  LEFT JOIN
    content c ON vh.content_id = c.content_id
  WHERE
    vh.profile_id = p_profile_id
    AND vh.percentage_watched < 90 -- Less than 90% watched
  ORDER BY
    vh.start_time DESC
  LIMIT p_limit;
END //
DELIMITER ;

CREATE OR REPLACE VIEW popular_content AS
SELECT
  c.content_id,
  c.title,
  c.type,
  c.poster_url,
  c.average_rating,
  c.total_ratings,
  COUNT(vh.history_id) AS view_count
FROM
  content c
LEFT JOIN
  viewing_history vh ON c.content_id = vh.content_id
GROUP BY
  c.content_id
ORDER BY
  view_count DESC, c.average_rating DESC;

DROP VIEW IF EXISTS trending_content;
CREATE VIEW trending_content AS
SELECT
  c.content_id,
  c.title,
  c.type,
  c.poster_url,
  c.average_rating,
  COUNT(vh.history_id) AS recent_views
FROM
  content c
JOIN
  viewing_history vh ON c.content_id = vh.content_id
WHERE
```

```
    vh.start_time > (CURRENT_DATE - INTERVAL 7 DAY)
GROUP BY
    c.content_id
ORDER BY
    recent_views DESC, c.average_rating DESC
LIMIT 50;

DELIMITER //
CREATE PROCEDURE get_subscription_stats()
BEGIN
    SELECT
        sp.name AS plan_name,
        COUNT(s.subscription_id) AS total_subscribers,
        SUM(CASE WHEN s.billing_cycle = 'monthly' THEN sp.price_monthly ELSE 0 END) AS monthly_revenue,
        SUM(CASE WHEN s.billing_cycle = 'yearly' THEN sp.price_yearly ELSE 0 END) AS yearly_revenue
    FROM
        subscription_plans sp
    LEFT JOIN
        subscriptions s ON sp.plan_id = s.plan_id AND s.status = 'active'
    GROUP BY
        sp.name
    ORDER BY
        total_subscribers DESC;
END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE get_user_activity_stats(IN p_days INTEGER)
BEGIN
    IF p_days IS NULL THEN
        SET p_days = 30;
    END IF;

    SELECT
        DATE(vh.start_time) AS activity_date,
        COUNT(DISTINCT vh.profile_id) AS active_users,
        SUM(vh.watch_duration_seconds) / 3600.0 AS total_watch_time_hours,
        AVG(vh.watch_duration_seconds) / 60.0 AS avg_session_minutes
    FROM
        viewing_history vh
    WHERE
        vh.start_time > (CURRENT_DATE - INTERVAL p_days DAY)
    GROUP BY
        DATE(vh.start_time)
    ORDER BY
        activity_date DESC;
END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE get_content_performance(IN p_limit INTEGER)
BEGIN
    IF p_limit IS NULL THEN
        SET p_limit = 20;
    END IF;

    SELECT
```

```
    c.content_id,
    c.title,
    c.type,
    COUNT(vh.history_id) AS total_views,
    AVG(vh.percentage_watched) AS completion_rate,
    c.average_rating,
    c.total_ratings
FROM
    content c
LEFT JOIN
    viewing_history vh ON c.content_id = vh.content_id
GROUP BY
    c.content_id
ORDER BY
    total_views DESC, completion_rate DESC
LIMIT p_limit;
```

- **ER DIAGRAM:**



📊 Charity Donation Tracker – Database Schema Overview

1. User & Profile Management

Table Name	Description
users	Registered users with authentication and personal details.
profiles	Multiple profiles per user, capturing preferences and activity.
admins	Platform administrators with role-based permissions.

Relationship: One-to-many between users and profiles.

---

2. Donation & Campaign Management

Table Name	Description
donors	Information about individuals or organizations making donations.
campaigns	Fundraising campaigns with goals, descriptions, and timelines.
donations	Records of individual donations linked to donors and campaigns.
donation_types	Categories of donations (e.g., monetary, in-kind).
donation_items	Details of in-kind donation items.

Relationships:

- One-to-many between donors and donations.

- **One-to-many between campaigns and donations.**
- **One-to-many between donation_types and donation_items.**

---

3. Distribution & Beneficiary Management

Table Name	Description
beneficiaries	Individuals or groups receiving aid.
distributions	Records of aid distributed to beneficiaries.
distribution_items	Specific items or services provided in each distribution.

***Relationships*:**

- **One-to-many between beneficiaries and distributions.**
- **One-to-many between distributions and distribution_items.**

---

4. User Interaction & Engagement

Table Name	Description
donation_feedback	Feedback from donors regarding their experience.
campaign_updates	Updates and progress reports for campaigns.
notifications	System-generated messages to users about campaigns and distributions.

***Relationships*:**

- **One-to-many between campaigns and campaign_updates.**

- **One-to-many between users and notifications.**

---

5. Financial & Transaction Management

Table Name	Description
payment_methods	Available payment options for donors.
transactions	Financial transactions linked to donations.
refunds	Records of any refunds processed.

Relationships:

- **One-to-many between donations and transactions.**
- **One-to-many between transactions and refunds.**

---

6. Analytics & Reporting

Table Name	Description
donation_statistics	Aggregated data on donations for reporting.
campaign_performance	Metrics evaluating campaign success.
beneficiary_impact	Data on the impact of distributions on beneficiaries.

Note: **These tables can be implemented as views or materialized views for performance optimization.**

---

7. Security & Access Control

Table Name	Description
roles	Different roles within the system (e.g., admin, donor, staff).
permissions	Specific permissions assigned to roles.
user_roles	Mapping of users to their roles.

Relationships:

- Many-to-many between users and roles through user_roles.
- One-to-many between roles and permissions.

---

8. System Maintenance & Logs

Table Name	Description
audit_logs	Records of system activities for auditing purposes.
error_logs	Logs of system errors and exceptions.
system_settings	Configuration settings for the application.

🔍 SQL QUERIES WITH OUTPUT (CHARITY DONATION TRACKER)

---

1. Get all donors who donated more than ₹500, ordered by amount

sql

CopyEdit

```
SELECT donor_id, donation_amount
FROM donations
```

```
WHERE donation_amount > 500
ORDER BY donation_amount DESC;
```

🟤 ***Output:***

donor_id	donation_amount
3	1000.00
1	500.00

---

2. Find total number of campaigns run by each charity

```
sql
CopyEdit
SELECT charity_id, COUNT(*) AS total_campaigns
FROM campaigns
GROUP BY charity_id;
```

🟤 ***Output:***

charity_id	total_campaigns
1	1
2	1
3	1

---

3. Top 5 campaigns based on donation amount received

```
sql
CopyEdit
SELECT c.campaign_id, c.campaign_name,
SUM(d.donation_amount) AS total_received
```

```
FROM donations d
JOIN campaigns c ON d.campaign_id = c.campaign_id
GROUP BY c.campaign_id, c.campaign_name
ORDER BY total_received DESC
LIMIT 5;
```

🔘 ***Output:***

campaign_id	campaign_name	total_received
3	Wildlife Protection Fund	1000.00
1	Disaster Relief Fund	500.00
2	Children Education Fund	200.00

---

4. Find all donors who donated to the 'Disaster Relief Fund'

sql

```
CopyEdit
SELECT d.donor_id, dn.name
FROM donations d
JOIN donors dn ON d.donor_id = dn.donor_id
JOIN campaigns c ON d.campaign_id = c.campaign_id
WHERE c.campaign_name = 'Disaster Relief Fund';
```

🔘 ***Output:***

donor_id	name
1	John Doe

---

5. Get average donation per campaign

sql

```
CopyEdit
SELECT campaign_id, AVG(donation_amount) AS
avg_donation
FROM donations
GROUP BY campaign_id;
```

🟤 ***Output:***

campaign_id	avg_donation
1	500.00
2	200.00
3	1000.00

---

6. List all donations that were refunded

sql

```
CopyEdit
SELECT d.donation_id, r.refund_amount, r.refund_reason
FROM donation_refunds r
JOIN donations d ON r.donation_id = d.donation_id;
```

🟤 ***Output:***

donation_id	refund_amount	refund_reason
1	50.00	Duplicate donation

---

7. Find all donors with recurring monthly donations

sql

```
CopyEdit
SELECT donor_id, amount
```

```
FROM recurring_donations
WHERE donation_frequency = 'monthly';
```

🟤 ***Output:***

donor_id	amount
1	100.00
2	50.00

---

8. Total funds allocated per campaign

```
sql
CopyEdit
SELECT campaign_id, SUM(allocated_amount) AS
total_allocated
FROM fund_allocation
GROUP BY campaign_id;
```

🟤 ***Output:***

campaign_id	total_allocated
1	5000.00
2	4000.00
3	7000.00

---

9. Count of social media shares by platform

```
sql
CopyEdit
SELECT platform, COUNT(*) AS total_shares
```

```
FROM social_media_shares
GROUP BY platform;
```

🟤 ***Output:***

platform	total_shares
facebook	1
twitter	1
instagram	1

---

10. List campaigns with updates

sql

```
CopyEdit
SELECT c.campaign_name, u.update_title, u.update_date
FROM campaign_updates u
JOIN campaigns c ON u.campaign_id = c.campaign_id;
```

🟤 ***Output:***

campaign_name	update_title	update_date
Disaster Relief Fund	Funds Raised	2025-01-20
Children Education Fund	Milestone Achieved	2025-02-15
Wildlife Protection Fund	Support Grows	2025-03-10

---

11. Get donor testimonials along with donor names

sql

```
CopyEdit
```

```
SELECT dt.testimonial_text, d.name
FROM donor_testimonials dt
JOIN donors d ON dt.donor_id = d.donor_id;
```

🟢 *Output:*

testimonial_text	name
I am proud to support the Disaster Relief Fund...	John Doe
Supporting education for children is important to me.	Jane Smith
I'm happy to support this cause.	Emily Johnson

---

12. List all donations eligible for tax deduction

```
sql
CopyEdit
SELECT t.donation_id, t.tax_deduction_percentage
FROM tax_deductible_donations t
WHERE t.eligible_for_tax_deduction = TRUE;
```

🟢 *Output:*

donation_id	tax_deduction_percentage
1	10.00
2	5.00
3	20.00

---

13. Get total gift items donated by each donor

```sql
CopyEdit
SELECT donor_id, SUM(quantity) AS total_gifts
FROM gift_donations
GROUP BY donor_id;
```

🟤 ***Output:***

donor_id	total_gifts
1	100
2	200
3	50

---

14. Show all events conducted by each charity

```sql
CopyEdit
SELECT c.name AS charity_name, e.event_name,
e.event_date
FROM fundraising_events e
JOIN charities c ON e.charity_id = c.charity_id;
```

🟤 ***Output:***

charity_name	event_name	event_date
Red Cross	Disaster Relief Gala	2025-01-25
Save the Children	Children's Education Auction	2025-02-20

charity_name	event_name	event_date
World Wildlife Fund	Wildlife Fundraiser	2025-03-15

---

15. Donors who donated but didn't receive a receipt

sql

CopyEdit

```
SELECT d.donor_id, d.donation_id
FROM donations d
LEFT JOIN donation_receipts r ON d.donation_id =
r.donation_id
WHERE r.receipt_id IS NULL;
```

🟤 ***Output:*** **(Assuming no missing receipts in current data, result may be empty.)**

☑ SUMMARY:

The Charity Donation Tracker is a comprehensive relational database system designed to manage and streamline charitable activities. It includes well-structured and normalized tables to handle donors, charities, campaigns, donations, events, beneficiaries, and fund distribution. Key features include tracking one-time and recurring donations, generating receipts, managing refunds, campaign updates, testimonials, and social media sharing. The database also supports fund allocation, gift donations, tax deductions, and analytics reports. Relationships between donors, campaigns, and charities are clearly defined to maintain data integrity and provide meaningful insights into donation impact and donor engagement.

☑ CONCLUSION:

The Charity Donation Tracker database offers a robust and scalable solution for managing end-to-end donation processes. With over 15 interlinked tables, it ensures efficient data management for tracking donations, distributing funds, organizing events, and analyzing donor behavior. The design supports both operational and analytical needs through well-structured SQL queries, making it suitable for real-world charity organizations. By incorporating features like recurring donations, refunds, beneficiary mapping, and tax deduction tracking, this database provides a complete framework for transparent and accountable charity management. It is optimized for performance and offers a solid foundation for building donation-driven platforms with clear visibility into every contribution.