**PostgreSQL Database Schema for Employee Scorecard Application**

**Document Version:** 1.0

**Date:** May 15, 2025

**Prepared By:** Amzad Basha

**1. roles Table**

This table will store the different user roles within the system.

CREATE TABLE roles (

id SERIAL PRIMARY KEY,

role\_name VARCHAR(50) UNIQUE NOT NULL -- e.g., 'Admin', 'Employee', 'Manager', 'Business Head'

);

-- Initial Data for roles (Example)

INSERT INTO roles (role\_name) VALUES

('Admin'),

('Employee'),

('Manager'),

('Business Head');

**2. users Table**

This table stores all user information, including a self-referencing foreign key for manager hierarchy.

CREATE TABLE users (

id SERIAL PRIMARY KEY,

employee\_id VARCHAR(50) UNIQUE NOT NULL, -- Company-specific unique ID for the employee

first\_name VARCHAR(100) NOT NULL,

last\_name VARCHAR(100) NOT NULL,

email VARCHAR(255) UNIQUE NOT NULL,

manager\_id INT REFERENCES users(id), -- Self-referencing FK to establish manager hierarchy

is\_active BOOLEAN DEFAULT TRUE NOT NULL,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP

);

-- Sample data

insert into users (employee\_id, first\_name, last\_name, email)

values ('EI-136', 'Ramesh', 'Maddukuri', 'ramesh.maddukuri@ispace.com');

insert into users (employee\_id, first\_name, last\_name, email, manager\_id)

values ('EI-135', 'Ramakrishna Rao', 'Bodi', 'ramakrishna.bodi@ispace.com', 1);

insert into users (employee\_id, first\_name, last\_name, email, manager\_id)

values ('EI-149', 'Amzad Basha', 'Shaik', 'amzadbasha.shaik@ispace.com', 2);

-- Index for faster lookups by manager

CREATE INDEX idx\_users\_manager\_id ON users(manager\_id);

-- Index for faster lookups by employee ID

CREATE INDEX idx\_users\_employee\_id ON users(employee\_id);

**3. user\_roles Table**

A junction table for a many-to-many relationship between users and roles, allowing a user to have multiple roles (though typically one primary role for this app).

CREATE TABLE user\_roles (

user\_id INT REFERENCES users(id) ON DELETE CASCADE,

role\_id INT REFERENCES roles(id) ON DELETE CASCADE,

PRIMARY KEY (user\_id, role\_id) -- Composite primary key

);

-- Sample Data

Insert into user\_roles(user\_id, role\_id) values (1, 4);

Insert into user\_roles(user\_id, role\_id) values (2, 3);

Insert into user\_roles(user\_id, role\_id) values (3, 2);

Insert into user\_roles(user\_id, role\_id) values (3, 1);

**4. activity\_types Table**

Defines the different types of contributions employees can make and their associated points.

CREATE TABLE activity\_types (

id SERIAL PRIMARY KEY,

name VARCHAR(100) UNIQUE NOT NULL, -- e.g., 'Code Commit', 'Code Review', 'Tech Talk Attended', 'Tech Talk Delivered'

points NUMERIC(10, 2) NOT NULL CHECK (points >= 0), -- Points awarded for this activity type

description TEXT,

is\_active BOOLEAN DEFAULT TRUE NOT NULL,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP

);

-- Example Data for activity\_types

INSERT INTO activity\_types (name, points, description) VALUES

('Code Contribution', 10.00, 'Contribution to code repository via commits/PRs'),

('Code Review', 5.00, 'Reviewed a peer''s code pull request'),

('Session Attended', 15.00, 'Attended an internal technical session or workshop'),

('Session Delivered', 50.00, 'Delivered an internal technical session or workshop'),

('Bug Fix (Critical)', 20.00, 'Fixed a critical production bug'),

('Feature Enhancement', 30.00, 'Implemented a new feature or significant enhancement');

**5. activity\_submissions Table**

Records each individual submission by an employee for an activity, along with its approval workflow status.

CREATE TYPE approval\_status AS ENUM (

'Pending Manager Approval',

'Approved by Manager',

'Pending Business Head Approval',

'Approved',

'Rejected'

);

CREATE TABLE activity\_submissions (

id SERIAL PRIMARY KEY,

employee\_id INT REFERENCES users(id) ON DELETE CASCADE NOT NULL,

activity\_type\_id INT REFERENCES activity\_types(id) ON DELETE RESTRICT NOT NULL, -- Don't allow deleting activity type if submissions exist

submission\_date DATE NOT NULL, -- The date the activity actually took place

description TEXT NOT NULL,

proof\_link VARCHAR(500), -- URL or direct link to proof (e.g., GitHub PR link, meeting recording)

-- Consider storing file paths if files are uploaded: proof\_file\_path VARCHAR(255),

submitted\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP NOT NULL,

current\_status approval\_status DEFAULT 'Pending Manager Approval' NOT NULL,

manager\_id\_approver INT REFERENCES users(id), -- Manager who approved/rejected

manager\_approval\_at TIMESTAMP WITH TIME ZONE,

manager\_comments TEXT,

business\_head\_id\_approver INT REFERENCES users(id), -- Business Head who approved/rejected

business\_head\_approval\_at TIMESTAMP WITH TIME ZONE,

business\_head\_comments TEXT,

awarded\_points NUMERIC(10, 2) DEFAULT 0.00 NOT NULL -- Final points awarded after full approval

);

-- Indexes for efficient querying by employee, activity type, and status

CREATE INDEX idx\_submissions\_employee\_id ON activity\_submissions(employee\_id);

CREATE INDEX idx\_submissions\_activity\_type\_id ON activity\_submissions(activity\_type\_id);

CREATE INDEX idx\_submissions\_status ON activity\_submissions(current\_status);

CREATE INDEX idx\_submissions\_manager\_approver ON activity\_submissions(manager\_id\_approver);

CREATE INDEX idx\_submissions\_business\_head\_approver ON activity\_submissions(business\_head\_id\_approver);

**6. notifications Table**

**(Optional but Recommended)** To store system notifications for users (e.g., submission status changes).

CREATE TABLE notifications (

id SERIAL PRIMARY KEY,

user\_id INT REFERENCES users(id) ON DELETE CASCADE NOT NULL,

message TEXT NOT NULL,

link VARCHAR(500), -- Link to the relevant submission or page

is\_read BOOLEAN DEFAULT FALSE NOT NULL,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP

);

CREATE INDEX idx\_notifications\_user\_id ON notifications(user\_id);

CREATE INDEX idx\_notifications\_is\_read ON notifications(is\_read);

**Explanation and Considerations:**

* **SERIAL for IDs:** Uses PostgreSQL's SERIAL data type, which auto-increments for primary keys.
* **NUMERIC(10, 2) for Points:** Good for storing monetary values or points that might have decimal places, preventing floating-point inaccuracies.
* **TEXT vs. VARCHAR:** TEXT is used for potentially longer strings like descriptions and comments, while VARCHAR(length) is used for fixed-length constraints like names or short IDs.
* **TIMESTAMP WITH TIME ZONE:** Recommended for created\_at, updated\_at, and approval timestamps to handle different time zones correctly.
* **ON DELETE CASCADE vs. ON DELETE RESTRICT:**
  + ON DELETE CASCADE (e.g., user\_roles): If a user is deleted, their entries in user\_roles are also deleted.
  + ON DELETE RESTRICT (e.g., activity\_submissions.activity\_type\_id): Prevents deletion of an activity\_type if there are still submissions linked to it, maintaining data integrity.
* **approval\_status ENUM:** Using an ENUM type is a clean way to manage the finite states of an approval workflow, making it type-safe and readable.
* **Indexes:** Added indexes on foreign key columns and frequently queried columns (current\_status, is\_read) to improve query performance.
* **awarded\_points:** Storing the final awarded\_points in activity\_submissions is crucial. While it could be derived from activity\_types.points on initial approval, storing it explicitly allows for potential future scenarios where points for a specific submission might be manually adjusted or differ from the default.
* **Calculated Scorecards:** The actual "scorecard" view for employees or managers would typically be a dynamic query (using SUM and GROUP BY) on the activity\_submissions table for approved entries. This avoids storing redundant data and ensures real-time accuracy. If performance becomes an issue with many users/submissions, you could consider a materialized view or a separate daily\_scores aggregation table.

This structure provides a solid foundation for your Employee Scorecard application.