Freelancer Mediator
SQL Queries and Operations

202203034 - Parth Sorathiya
202203065 - Paawan Vala

Contents

1 Table Creation(DDL)

```
-- Create enums for statuses
CREATE TYPE project_status AS ENUM ('open', 'in_progress', 'completed',
   → 'cancelled');
CREATE TYPE contract_status AS ENUM ('active', 'completed', 'terminated
   → ');
CREATE TYPE proposal_status AS ENUM ('pending', 'accepted', 'rejected')
   → ;

-- Users
CREATE TABLE users (
  user_id       SERIAL PRIMARY KEY,
  name          VARCHAR(255)   NOT NULL,
  email         VARCHAR(255)   NOT NULL UNIQUE,
  password      VARCHAR(255)   NOT NULL,
  dob           DATE,
  country       VARCHAR(100),
  created_at    TIMESTAMP      NOT NULL DEFAULT NOW(),
  updated_at    TIMESTAMP      NOT NULL DEFAULT NOW()
);

-- Freelancers
CREATE TABLE freelancers (
  freelancer_id         SERIAL PRIMARY KEY,
  user_id               INTEGER NOT NULL
                          REFERENCES users(user_id)
                          ON DELETE CASCADE,
  description           TEXT,
  experience            INTEGER      DEFAULT 0,
  rating                NUMERIC(3,2) DEFAULT 0.00,
  availability_status   BOOLEAN      DEFAULT TRUE
);

-- Clients
CREATE TABLE clients (
  client_id     SERIAL PRIMARY KEY,
  user_id       INTEGER NOT NULL
                  REFERENCES users(user_id)
                  ON DELETE CASCADE,
  company_name VARCHAR(255),
  description   TEXT,
  rating        NUMERIC(3,2) DEFAULT 0.00
);

-- Skills
CREATE TABLE skills (
  skill_id     SERIAL PRIMARY KEY,
  skill_name   VARCHAR(100) NOT NULL UNIQUE
);

-- Freelancer_skills (M-N between freelancers & skills)
CREATE TABLE freelancer_skills (
  freelancer_id   INTEGER NOT NULL
                    REFERENCES freelancers(freelancer_id)
                    ON DELETE CASCADE,
  skill_id        INTEGER NOT NULL
```

```sql
                    REFERENCES skills(skill_id)
                    ON DELETE CASCADE,
  PRIMARY KEY (freelancer_id, skill_id)
);

-- Projects
CREATE TABLE projects (
  project_id   SERIAL PRIMARY KEY,
  client_id    INTEGER NOT NULL
                  REFERENCES clients(client_id)
                  ON DELETE CASCADE,
  title        VARCHAR(255)   NOT NULL,
  description  TEXT,
  budget       NUMERIC(12,2),
  deadline     DATE,
  status       project_status NOT NULL DEFAULT 'open',
  created_at   TIMESTAMP      NOT NULL DEFAULT NOW(),
  updated_at   TIMESTAMP      NOT NULL DEFAULT NOW()
);

-- Required_skills (M-N between projects & skills)
CREATE TABLE required_skills (
  project_id  INTEGER NOT NULL
                  REFERENCES projects(project_id)
                  ON DELETE CASCADE,
  skill_id    INTEGER NOT NULL
                  REFERENCES skills(skill_id)
                  ON DELETE CASCADE,
  PRIMARY KEY (project_id, skill_id)
);

-- Contracts
CREATE TABLE contracts (
  contract_id   SERIAL PRIMARY KEY,
  project_id    INTEGER NOT NULL
                   REFERENCES projects(project_id)
                   ON DELETE CASCADE,
  freelancer_id INTEGER NOT NULL
                   REFERENCES freelancers(freelancer_id)
                   ON DELETE CASCADE,
  start_date    DATE,
  end_date      DATE,
  amount        NUMERIC(12,2),
  status        contract_status NOT NULL DEFAULT 'active',
  created_at    TIMESTAMP       NOT NULL DEFAULT NOW(),
  updated_at    TIMESTAMP       NOT NULL DEFAULT NOW(),
  UNIQUE (project_id, freelancer_id)
);

-- Reviews
CREATE TABLE reviews (
  review_id    SERIAL PRIMARY KEY,
  contract_id  INTEGER NOT NULL
                   REFERENCES contracts(contract_id)
                   ON DELETE CASCADE,
  reviewer_id  INTEGER NOT NULL
                   REFERENCES users(user_id)
                   ON DELETE SET NULL,
  reviewee_id  INTEGER NOT NULL
                   REFERENCES users(user_id)
                   ON DELETE SET NULL,
  rating       INTEGER CHECK (rating BETWEEN 1 AND 5),
  feedback     TEXT,
  created_at   TIMESTAMP NOT NULL DEFAULT NOW(),
  updated_at   TIMESTAMP NOT NULL DEFAULT NOW()
);

-- Proposals
CREATE TABLE proposals (
  project_id    INTEGER NOT NULL
                   REFERENCES projects(project_id)
                   ON DELETE CASCADE,
  freelancer_id INTEGER NOT NULL
                   REFERENCES freelancers(freelancer_id)
                   ON DELETE CASCADE,
  proposal      TEXT,
  bid_amount    NUMERIC(12,2),
  status        proposal_status NOT NULL DEFAULT 'pending',
  created_at    TIMESTAMP NOT NULL DEFAULT NOW(),
  PRIMARY KEY (project_id, freelancer_id)
);

-- Payments
CREATE TABLE payments (
  contract_id  INTEGER PRIMARY KEY
                   REFERENCES contracts(contract_id)
                   ON DELETE CASCADE,
  amount       NUMERIC(12,2) NOT NULL,
  payment_at   TIMESTAMP     NOT NULL DEFAULT NOW()
);
```

2 CRUD Operations for All Tables

This section provides basic Create, Read, Update, and Delete operations for each table in the schema.

2.1 Users

```
-- Insert a new user into the Users table
INSERT INTO Users (name, email, password, dob, country, created_at,
   → updated_at)
VALUES ('Alice', 'alice@example.com', 'securePass!', '1990-05-14', '
   → India', NOW(), NOW());

-- Retrieve a user by their ID
SELECT *
FROM Users
WHERE user_id = 1;

-- Update a user's name and timestamp
UPDATE Users
SET name = 'Alice Smith', updated_at = NOW()
```

```
WHERE user_id = 1;

-- Delete a user by their ID
DELETE
FROM Users
WHERE user_id = 1;
```

we can follow the same structure for other tables like Clients, Freelancers, Skills, Projects, Proposals, Contracts, Payments, Reviews, Freelancer$_skills$, $Required_skills$

3 User Flow–Based Queries

3.1 User Information and Identification

Calculate a user's age in years. Uses PostgreSQL's AGE() function to find the interval between today and the stored date of birth, then extracts the year component for a clear age value.

```
-- Calculate age in years for user with ID = 3
SELECT EXTRACT(YEAR FROM AGE(CURRENT_DATE, dob)) AS Age
FROM Users
WHERE user_id = 3;
```

Find the user_id associated with a given freelancer. Each freelancer record references exactly one user; this query retrieves that link.

```
-- Retrieve user_id for freelancer with ID = 3
SELECT user_id
FROM Freelancers
WHERE freelancer_id = 3;
```

Find the user_id associated with a given client. Similar to the freelancer lookup, useful when transitioning from client-specific tables back to user account details.

```
-- Retrieve user_id for client with ID = 4
SELECT user_id
FROM Clients
WHERE client_id = 4;
```

Obtain a freelancer's name via their freelancer_id. Demonstrates a subquery that maps freelancer_id → user_id → name in two steps.

```
-- Get the name of the freelancer whose ID is 3
SELECT name
FROM Users
WHERE user_id = (
    SELECT user_id
    FROM Freelancers
    WHERE freelancer_id = 3
);
```

3.2 Skill Lookup

List all skills possessed by a specific freelancer. Joins the Skills master table with Freelancer_skills to fetch skill names for the target freelancer.

```
-- List skill names for freelancer with ID = 3
SELECT s.skill_name
FROM Skills AS s
INNER JOIN (
    SELECT *
    FROM Freelancer_skills
    WHERE freelancer_id = 3
) AS fs ON s.skill_id = fs.skill_id;
```

List all skills required by a specific project. Same join pattern, but on the Required_skills side to show project requirements.

```
-- List skill names required for project with ID = 1
SELECT s.skill_name
FROM Skills AS s
INNER JOIN (
    SELECT *
    FROM Required_skills
    WHERE project_id = 1
) AS rs ON s.skill_id = rs.skill_id;
```

3.3 Reviews and Ratings

Compute the average rating received by a user. Aggregates the ratings for a given reviewee_id, rounds to two decimal places.

```
-- Calculate average rating for user with ID = 3
SELECT ROUND(AVG(rating),2) AS average_rating
FROM Reviews
WHERE reviewee_id = 3
GROUP BY reviewee_id;
```

Retrieve all reviews about a specific user. Fetches every row in Reviews where the user is the reviewee.

```
-- Get all reviews for user with ID = 3
SELECT *
FROM Reviews
WHERE reviewee_id = 3;
```

3.4 Project Matching and Bidding

Find open projects matching at least one of a freelancer's skills. Uses an EXISTS clause with INTERSECT to detect any skill overlap, and filters by budget and active deadline.

```
-- Find open projects that match at least one skill of freelancer with
    → ID = 1
SELECT *
FROM Projects AS p
WHERE p.status = 'open'
  AND EXISTS (
      (SELECT skill_id FROM Required_skills WHERE project_id = p.
          → project_id)
      INTERSECT
      (SELECT skill_id FROM Freelancer_skills WHERE freelancer_id = 1)
  )
  AND p.budget >= 0
  AND CURRENT_DATE <= p.deadline;
```

Retrieve open projects for which a freelancer meets all required skills. The `NOT EXISTS` pattern ensures the freelancer has no missing required skills.

```
-- Get all open projects for which freelancer with ID = 1 has every
   → required skill
SELECT p.*
FROM Projects AS p
WHERE p.status = 'open'
  AND CURRENT_DATE <= p.deadline
  AND NOT EXISTS (
      SELECT 1
      FROM Required_skills AS rs
      WHERE rs.project_id = p.project_id
        AND rs.skill_id NOT IN (
            SELECT fs.skill_id
            FROM Freelancer_skills AS fs
            WHERE fs.freelancer_id = 1
        )
  );
```

List proposals where the freelancer possesses every skill required by that project. Combines proposals filtering with a complete skill-match check.

```
-- Select proposals for project 1 where the bidder has all required
   → skills
SELECT *
FROM Proposals AS p
WHERE p.project_id = 1
  AND NOT EXISTS (
      SELECT 1
      FROM Required_skills AS rs
      WHERE rs.project_id = 1
        AND rs.skill_id NOT IN (
            SELECT fs.skill_id
            FROM Freelancer_skills AS fs
            WHERE fs.freelancer_id = p.freelancer_id
        )
  );
```

3.5 Proposals and Filtering

Fetch all open proposals for a client's active projects. Inner query finds the client's open projects; outer query returns associated proposals.

```
-- Get all proposals for client 4 s  currently open projects
SELECT *
FROM Proposals
WHERE project_id IN (
    SELECT project_id
    FROM Projects
    WHERE client_id = 4
      AND status = 'open'
);
```

Retrieve every proposal submitted to a specific project. Direct lookup by `project_id`.

```sql
-- List all proposals for project with ID = 6
SELECT *
FROM Proposals
WHERE project_id = 6;
```

Order a project's proposals by the freelancer's rating (highest first). Joins to the `Freelancers` table and sorts descending.

```sql
-- List proposals for project 6 sorted by freelancer rating (descending
   → )
SELECT f.freelancer_id,
       p.proposal,
       f.rating
FROM (
    SELECT *
    FROM Proposals
    WHERE project_id = 6
) AS p
INNER JOIN Freelancers AS f
  ON p.freelancer_id = f.freelancer_id
ORDER BY f.rating DESC;
```

Order a project's proposals by bid amount (lowest first). Shows the most cost-effective bids at the top, facilitating quick client decision-making.

```sql
-- List proposals for project 6 sorted by bid amount (ascending)
SELECT f.freelancer_id,
       p.proposal,
       p.bid_amount
FROM (
    SELECT *
    FROM Proposals
    WHERE project_id = 6
) AS p
INNER JOIN Freelancers AS f
  ON p.freelancer_id = f.freelancer_id
ORDER BY p.bid_amount ASC;
```

Filter a project's proposals to those within the project's budget, sorted by bid. Joins with the `Projects` table to compare bid against the project's budget.

```sql
-- List proposals for project 6 where bid_amount     project budget,
   → sorted by bid (ascending)
SELECT f.freelancer_id,
       p.proposal,
       p.bid_amount,
       p.*
FROM (
    SELECT *
    FROM Proposals
    WHERE project_id = 6
) AS p
INNER JOIN Freelancers AS f
```

```
  ON p.freelancer_id = f.freelancer_id
INNER JOIN Projects AS proj
  ON p.project_id = proj.project_id
WHERE p.bid_amount <= proj.budget
ORDER BY p.bid_amount ASC;
```

Order a project's proposals by the freelancer's experience (highest first). Helps clients identify the most seasoned bidders.

```
-- List proposals for project 6 sorted by freelancer experience (
    → descending)
SELECT f.freelancer_id,
       p.proposal,
       f.experience
FROM (
    SELECT *
    FROM Proposals
    WHERE project_id = 6
) AS p
INNER JOIN Freelancers AS f
  ON p.freelancer_id = f.freelancer_id
ORDER BY f.experience DESC;
```

3.6 Earnings and Payments

List every payment received by a freelancer. A Common Table Expression (CTE) gathers the freelancer's contracts; then the main query retrieves all matching payments.

```
-- Retrieve all payments for freelancer with ID = 3
WITH MyContracts AS (
    SELECT contract_id
    FROM Contracts
    WHERE freelancer_id = 3
)
SELECT *
FROM Payments
WHERE contract_id IN (SELECT contract_id FROM MyContracts);
```

Calculate the total earnings for a freelancer across all contracts.

```
-- Sum total payment amounts for freelancer with ID = 3
WITH MyContracts AS (
    SELECT contract_id
    FROM Contracts
    WHERE freelancer_id = 3
)
SELECT SUM(amount) AS total_earnings
FROM Payments
WHERE contract_id IN (SELECT contract_id FROM MyContracts);
```

Break down a freelancer's earnings by year and month.

```
-- Compute monthly earnings for freelancer with ID = 3
WITH MyContracts AS (
    SELECT contract_id
    FROM Contracts
    WHERE freelancer_id = 3
)
SELECT
    EXTRACT(YEAR FROM payment_at)  AS year,
    EXTRACT(MONTH FROM payment_at) AS month,
    SUM(amount)                    AS monthly_total
FROM Payments
WHERE contract_id IN (SELECT contract_id FROM MyContracts)
GROUP BY year, month;
```

Provide a year-wise earnings summary for a freelancer.

```
-- Compute yearly earnings for freelancer with ID = 3
WITH MyContracts AS (
    SELECT contract_id
    FROM Contracts
    WHERE freelancer_id = 3
)
SELECT
    EXTRACT(YEAR FROM payment_at) AS year,
    SUM(amount)                   AS yearly_total
FROM Payments
WHERE contract_id IN (SELECT contract_id FROM MyContracts)
GROUP BY year;
```

Identify all active contracts without a recorded payment.

```
-- List contracts for freelancer 3 that have no corresponding payment
SELECT *
FROM Contracts
WHERE freelancer_id = 3
  AND contract_id NOT IN (
      SELECT contract_id
      FROM Payments
  );
```

3.7 Analytics and Insights

Generate a leaderboard of freelancers by earnings over the last 30 days. Uses a date filter to limit to recent payments, aggregates by freelancer, and sorts descending.

```
-- Top earning freelancers in the last 30 days
SELECT
    p.freelancer_id,
    SUM(p.amount) AS total_earning
FROM Payments AS p
NATURAL JOIN Contracts AS c
WHERE p.payment_at >= (CURRENT_DATE - INTERVAL '30 days')
GROUP BY p.freelancer_id
ORDER BY total_earning DESC;
```

Count how many projects a client has created in the past year.

```
-- Count of projects created by client 1 in the last 12 months
SELECT
    client_id,
    COUNT(*) AS total_projects
```

```sql
FROM Projects
WHERE client_id = 1
  AND created_at >= (CURRENT_DATE - INTERVAL '1 year')
GROUP BY client_id;
```

List the top five most frequently required skills across all projects.

```sql
-- Top five skills by number of projects requiring them
SELECT
    s.skill_name AS skill,
    COUNT(*)      AS usage_count
FROM Required_skills AS rs
NATURAL JOIN Skills AS s
GROUP BY s.skill_id, s.skill_name
ORDER BY usage_count DESC
LIMIT 5;
```

List the top ten skills most commonly listed by freelancers.

```sql
-- Top ten freelancer skills by listing frequency
SELECT
    s.skill_name AS skill,
    COUNT(*)      AS listing_count
FROM Freelancer_skills AS fs
NATURAL JOIN Skills AS s
GROUP BY s.skill_id, s.skill_name
ORDER BY listing_count DESC
LIMIT 10;
```

Identify the top five skills generating the highest total contract revenue.

```sql
-- Top five skills by total revenue from associated contracts
SELECT
    s.skill_name AS skill,
    SUM(c.amount) AS total_revenue
FROM Contracts AS c
NATURAL JOIN Required_skills AS rs
NATURAL JOIN Skills AS s
GROUP BY s.skill_id, s.skill_name
ORDER BY total_revenue DESC
LIMIT 5;
```

4 Conclusion

This document supplements the main DBMS report by providing detailed SQL queries organized by user workflow and augmented with clear descriptions to explain intent and data flow.