QPerform Database Structure Documentation

Version: 1.0 **Last Updated:** December 9, 2025 **Database:** PostgreSQL **Application:** QPerform Performance Management System

Table of Contents

Database Overview

Database Information

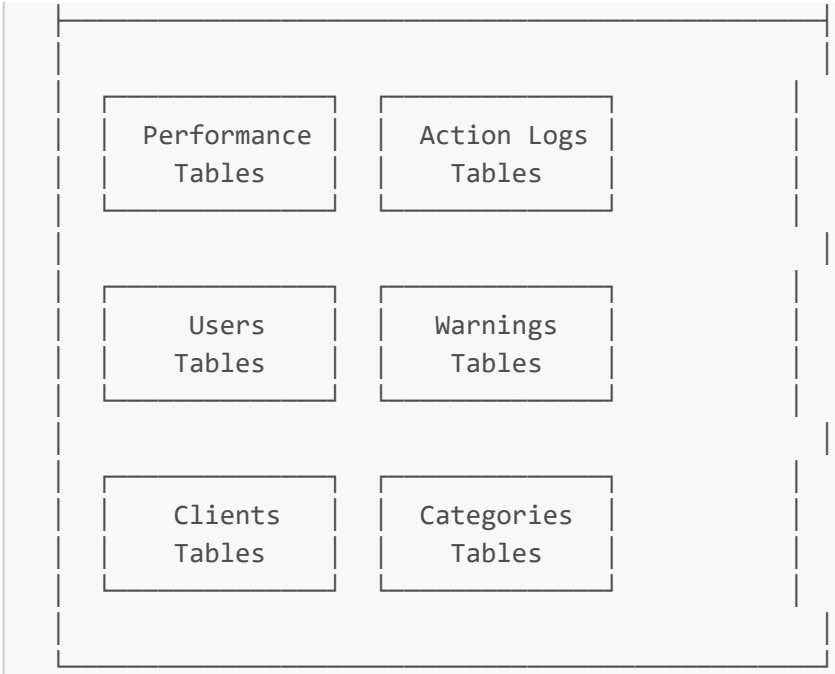
- **Database Name:** `qperform_db`
- **Database System:** PostgreSQL 14+
- **Character Set:** UTF8
- **Collation:** en_US.UTF-8
- **Timezone:** UTC

Purpose

The QPerform database stores:

- Agent performance metrics (QA and Production)
- Action logs (warnings, coaching, etc.)
- User profiles and permissions
- Warning tracking and expiration
- Historical performance data
- Monthly summary aggregations

Database Architecture



Schema Design

Core Schemas

1. `public` Schema

Contains all application tables and views

2. Future Schemas (Planned)

- `audit` - Audit trail and change history
- `reporting` - Pre-aggregated reporting data
- `archive` - Historical data beyond retention period

Tables

1. performance_data

Purpose: Stores weekly performance metrics for each agent

Structure:

```
CREATE TABLE performance_data (
    id SERIAL PRIMARY KEY,
    agent_email VARCHAR(255) NOT NULL,
    agent_id VARCHAR(100),
    agent_name VARCHAR(255),
    position VARCHAR(100),
    office VARCHAR(100),
```

```
    client VARCHAR(255) NOT NULL,
    task VARCHAR(255),
    category VARCHAR(255) NOT NULL,

    -- QA Metrics
    kpi_qa DECIMAL(5,2) NOT NULL,
    flag_qa VARCHAR(50),

    -- Production Metrics
    kpi_avg_prod DECIMAL(5,2) NOT NULL,
    flag_prod VARCHAR(50),

    -- Week Information
    week_range VARCHAR(50) NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,

    -- Month/Year Information
    month_num INTEGER NOT NULL,
    month_name VARCHAR(20) NOT NULL,
    year_num INTEGER NOT NULL,

    -- Timestamps
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    -- Constraints
    CONSTRAINT valid_kpi_qa CHECK (kpi_qa >= 0 AND kpi_qa <= 200),
    CONSTRAINT valid_kpi_prod CHECK (kpi_avg_prod >= 0 AND kpi_avg_prod <= 200),
    CONSTRAINT valid_month CHECK (month_num >= 1 AND month_num <= 12),
    CONSTRAINT valid_year CHECK (year_num >= 2020)
);
```

Indexes:

```
CREATE INDEX idx_perf_agent_email ON performance_data(agent_email);
CREATE INDEX idx_perf_client ON performance_data(client);
CREATE INDEX idx_perf_dates ON performance_data(start_date, end_date);
CREATE INDEX idx_perf_month_year ON performance_data(month_num, year_num);
CREATE INDEX idx_perf_week_range ON performance_data(week_range);
CREATE INDEX idx_perf_category ON performance_data(category);
```

Sample Data:

```
id | agent_email           | agent_name   | client      | category     | kpi_qa |
kpi_avg_prod | week_range      | start_date | end_date   | month_name | year_num
---|-----------------------|--------------|-------------|--------------|--------|-
-------------|-----------------|------------|------------|------------|----------
1  | john.doe@onq.com      | John Doe     | Client A    | Data Entry   | 98.5   |
```

```
102.1         | 12/01 - 12/07    | 2025-12-01 | 2025-12-07 | December    | 2025
2   | jane.smith@onq.com     | Jane Smith    | Client B     | QA Review     | 96.2    |
99.8          | 12/01 - 12/07    | 2025-12-01 | 2025-12-07 | December    | 2025
```

---

2. action_logs

Purpose: Stores all corrective actions taken against agents

Structure:

```sql
CREATE TABLE action_logs (
    id SERIAL PRIMARY KEY,
    agent_email VARCHAR(255) NOT NULL,
    agent_name VARCHAR(255),
    action_date DATE NOT NULL,
    action_type VARCHAR(100) NOT NULL,
    description TEXT NOT NULL,
    taken_by VARCHAR(255) NOT NULL,

    -- Week Information
    week_start_date DATE,
    week_end_date DATE,

    -- Client/Category
    client VARCHAR(255),
    category VARCHAR(255),

    -- Warning Information
    warning_type VARCHAR(100),
    expiration_date DATE,
    is_active BOOLEAN DEFAULT true,

    -- Timestamps
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    -- Constraints
    CONSTRAINT valid_action_type CHECK (
        action_type IN (
            'Coaching',
            'Training',
            'Counseling',
            'Verbal Warning',
            'Written Warning',
            'Final Warning',
            'PIP',
            'Termination',
            'Other'
        )
    ),
    CONSTRAINT valid_warning_type CHECK (
```

```
        warning_type IN (
            'Substandard Work - Production',
            'Substandard Work - QA',
            'Other',
            NULL
        )
    ),
    CONSTRAINT description_length CHECK (LENGTH(description) >= 10)
);
```

Indexes:

```
CREATE INDEX idx_action_agent_email ON action_logs(agent_email);
CREATE INDEX idx_action_type ON action_logs(action_type);
CREATE INDEX idx_action_date ON action_logs(action_date);
CREATE INDEX idx_action_expiration ON action_logs(expiration_date);
CREATE INDEX idx_action_active ON action_logs(is_active);
CREATE INDEX idx_action_warning_type ON action_logs(warning_type);
CREATE INDEX idx_action_week ON action_logs(week_start_date, week_end_date);
```

Sample Data:

```
id | agent_email        | action_date | action_type      | warning_type
| expiration_date | is_active | description
---|--------------------| ------------|------------------|-------------------------
---------|-----------------|-----------|-------------
1  | john.doe@onq.com   | 2025-12-01  | Verbal Warning   | Substandard Work - QA
| 2026-03-01      | true      | Verbal warning issued for QA performance below
97%...
2  | jane.smith@onq.com | 2025-11-15  | Written Warning  | Substandard Work -
Production   | 2026-05-14      | true      | Written warning issued after 2 verbal
warnings...
```

---

3. users

Purpose: Stores user profiles and authentication information

Structure:

```
CREATE TABLE users (
    id VARCHAR(100) PRIMARY KEY,  -- Microsoft Azure AD Object ID
    email VARCHAR(255) UNIQUE NOT NULL,
    display_name VARCHAR(255) NOT NULL,
    job_title VARCHAR(255),
    department VARCHAR(255),
    office VARCHAR(100),
```

```
    -- Role & Permissions
    role VARCHAR(50) NOT NULL,
    assigned_clients TEXT[],  -- Array of client names

    -- Authentication
    last_login TIMESTAMP,
    is_active BOOLEAN DEFAULT true,

    -- Timestamps
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    -- Constraints
    CONSTRAINT valid_role CHECK (
        role IN (
            'AVP',
            'Director',
            'Manager',
            'Assistant Manager',
            'Supervisor',
            'Team Lead',
            'Agent',
            'Unauthorized'
        )
    )
);
```

Indexes:

```
CREATE INDEX idx_user_email ON users(email);
CREATE INDEX idx_user_role ON users(role);
CREATE INDEX idx_user_active ON users(is_active);
```

Sample Data:

```
id                                   | email                    | display_name
| role     | assigned_clients
-------------------------------------|--------------------------|---------------
--|----------|------------------
a1b2c3d4-e5f6-7890-abcd-ef1234567890 | sarah.johnson@onq.com    | Sarah Johnson
| Manager  | {"Client A", "Client B"}
b2c3d4e5-f6g7-8901-bcde-fg2345678901 | mike.brown@onq.com       | Mike Brown
| Director | NULL (all clients)
```

---

4. clients

Purpose: Stores client information and metadata

Structure:

```
CREATE TABLE clients (
    id SERIAL PRIMARY KEY,
    client_name VARCHAR(255) UNIQUE NOT NULL,
    client_code VARCHAR(50),
    industry VARCHAR(100),
    start_date DATE,
    end_date DATE,
    is_active BOOLEAN DEFAULT true,

    -- Contact Information
    primary_contact VARCHAR(255),
    contact_email VARCHAR(255),

    -- Metadata
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Sample Data:

```
id | client_name  | client_code | is_active | primary_contact
---|--------------|-------------|-----------|-----------------
1  | Client A     | CLA         | true      | contact@clienta.com
2  | Client B     | CLB         | true      | contact@clientb.com
```

---

5. categories

Purpose: Stores performance categories/task types

Structure:

```
CREATE TABLE categories (
    id SERIAL PRIMARY KEY,
    category_name VARCHAR(255) UNIQUE NOT NULL,
    category_code VARCHAR(50),
    description TEXT,
    client_id INTEGER REFERENCES clients(id),

    -- Thresholds
    qa_critical_threshold DECIMAL(5,2) DEFAULT 97.00,
    prod_critical_threshold DECIMAL(5,2) DEFAULT 98.00,

    is_active BOOLEAN DEFAULT true,
```

```
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Sample Data:

```
id | category_name | client_id | qa_critical_threshold | prod_critical_threshold
---|---------------|-----------|-----------------------|------------------------
1  | Data Entry    | 1         | 97.00                 | 98.00
2  | QA Review     | 1         | 97.00                 | 98.00
3  | Customer Svc  | 2         | 97.00                 | 98.00
```

---

6. warning_templates

Purpose: Stores warning letter templates

Structure:

```
CREATE TABLE warning_templates (
    id SERIAL PRIMARY KEY,
    warning_type VARCHAR(100) NOT NULL,
    template_name VARCHAR(255) NOT NULL,
    subject_line VARCHAR(500),
    body_template TEXT NOT NULL,

    -- Variables that can be used: {agent_name}, {date}, {reason}, etc.
    available_variables TEXT[],

    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT unique_template_type UNIQUE(warning_type, template_name)
);
```

---

7. monthly_summaries

Purpose: Pre-aggregated monthly summary data for performance

Structure:

```
CREATE TABLE monthly_summaries (
    id SERIAL PRIMARY KEY,
    month_num INTEGER NOT NULL,
    year_num INTEGER NOT NULL,
```

```
    client VARCHAR(255) NOT NULL,
    category VARCHAR(255) NOT NULL,

    -- Aggregated Metrics
    total_aftes INTEGER DEFAULT 0,
    underperformers INTEGER DEFAULT 0,
    weeks_with_issues INTEGER DEFAULT 0,
    avg_score DECIMAL(5,2),

    -- QA vs Production
    avg_qa_score DECIMAL(5,2),
    avg_prod_score DECIMAL(5,2),

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT unique_summary UNIQUE(month_num, year_num, client, category)
);
```

Indexes:

```
CREATE INDEX idx_summary_month_year ON monthly_summaries(month_num, year_num);
CREATE INDEX idx_summary_client ON monthly_summaries(client);
```

---

8. agent_warnings (View - Computed)

Purpose: Real-time view of active warnings per agent

Structure:

```
CREATE OR REPLACE VIEW agent_warnings AS
SELECT
    agent_email,
    warning_type,
    COUNT(*) FILTER (WHERE action_type = 'Verbal Warning' AND is_active = true) as
verbal_warnings,
    COUNT(*) FILTER (WHERE action_type = 'Written Warning' AND is_active = true)
as written_warnings,
    COUNT(*) FILTER (WHERE action_type = 'Final Warning' AND is_active = true) as
final_warnings,
    COUNT(*) FILTER (WHERE action_type = 'PIP' AND is_active = true) as pip_count,
    MAX(action_date) FILTER (WHERE action_type = 'Verbal Warning' AND is_active =
true) as last_verbal_date,
    MAX(action_date) FILTER (WHERE action_type = 'Written Warning' AND is_active =
true) as last_written_date,
    MIN(expiration_date) FILTER (WHERE is_active = true) as earliest_expiration
FROM action_logs
WHERE is_active = true
GROUP BY agent_email, warning_type;
```

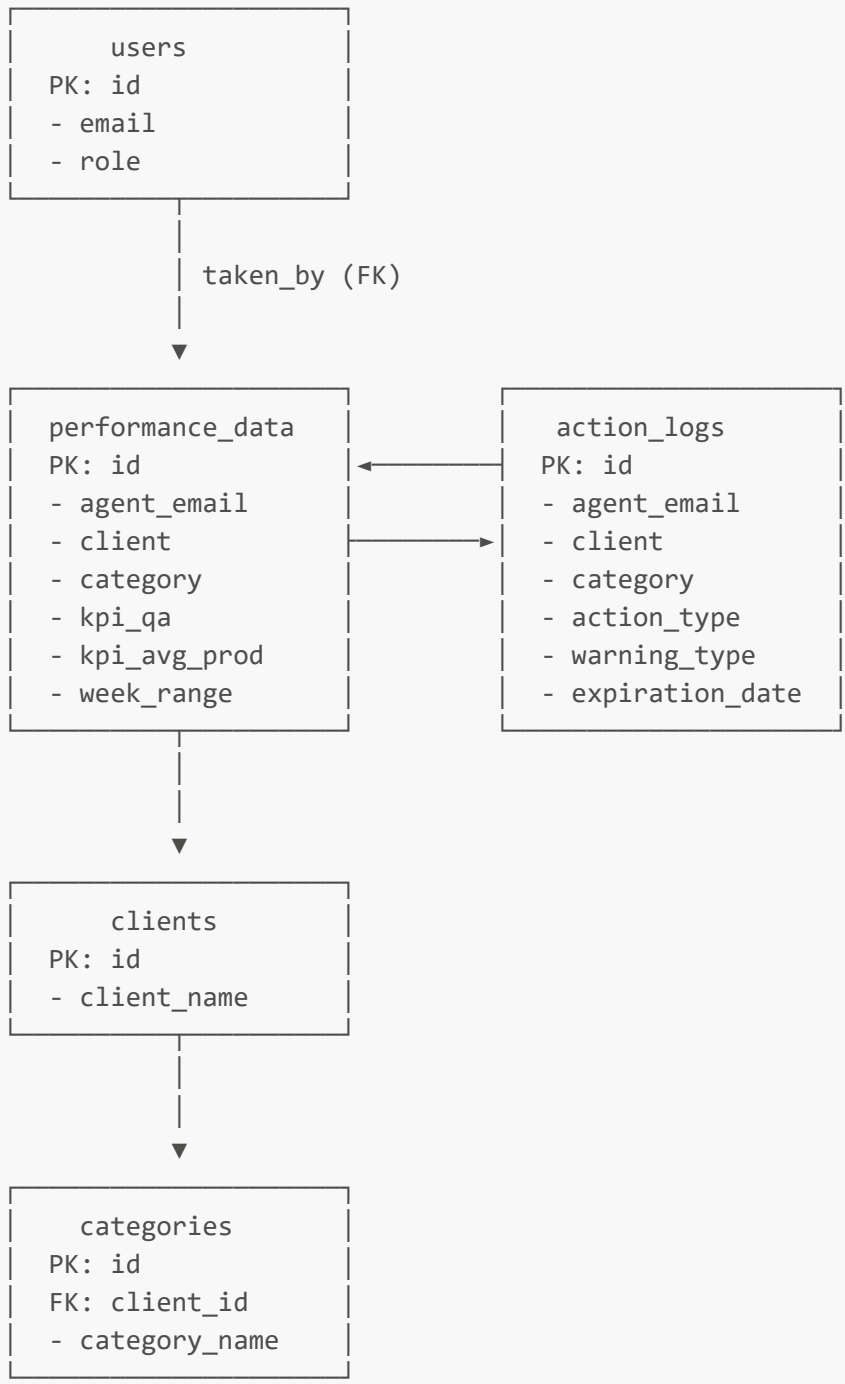
Data Types

Custom Types (Enums)

```
-- Action Types
CREATE TYPE action_type_enum AS ENUM (
    'Coaching',
    'Training',
    'Counseling',
    'Verbal Warning',
    'Written Warning',
    'Final Warning',
    'PIP',
    'Termination',
    'Other'
);

-- Warning Categories
CREATE TYPE warning_category_enum AS ENUM (
    'Substandard Work - Production',
    'Substandard Work - QA',
    'Other'
);

-- User Roles
CREATE TYPE user_role_enum AS ENUM (
    'AVP',
    'Director',
    'Manager',
    'Assistant Manager',
    'Supervisor',
    'Team Lead',
    'Agent',
    'Unauthorized'
);

-- Performance Flags
CREATE TYPE performance_flag_enum AS ENUM (
    'Critical',
    'Low',
    'Normal',
    'Good',
    'Great'
);
```

Relationships

Entity Relationship Diagram



Foreign Key Relationships

```sql
-- Link categories to clients
ALTER TABLE categories
ADD CONSTRAINT fk_category_client
FOREIGN KEY (client_id) REFERENCES clients(id)
ON DELETE CASCADE;

-- Optional: Link performance_data to users
ALTER TABLE performance_data
ADD CONSTRAINT fk_performance_agent
FOREIGN KEY (agent_email) REFERENCES users(email)
```

```
ON DELETE CASCADE;

-- Optional: Link action_logs to users
ALTER TABLE action_logs
ADD CONSTRAINT fk_action_agent
FOREIGN KEY (agent_email) REFERENCES users(email)
ON DELETE CASCADE;
```

Indexes

Performance Optimization Indexes

```
-- Composite indexes for common queries
CREATE INDEX idx_perf_composite_1 ON performance_data(client, month_num,
year_num);
CREATE INDEX idx_perf_composite_2 ON performance_data(agent_email, start_date,
end_date);
CREATE INDEX idx_action_composite_1 ON action_logs(agent_email, is_active,
expiration_date);
CREATE INDEX idx_action_composite_2 ON action_logs(action_type, warning_type,
is_active);

-- Full-text search indexes
CREATE INDEX idx_action_description_fts ON action_logs USING
GIN(to_tsvector('english', description));

-- Partial indexes for common filters
CREATE INDEX idx_active_warnings ON action_logs(agent_email, warning_type)
WHERE is_active = true;

CREATE INDEX idx_critical_performance ON performance_data(agent_email)
WHERE kpi_qa < 97 OR kpi_avg_prod < 98;
```

Views

1. v_underperforming_agents

Purpose: Quick view of agents currently underperforming

```
CREATE OR REPLACE VIEW v_underperforming_agents AS
SELECT
    p.agent_email,
    p.agent_name,
    p.client,
    p.category,
    p.week_range,
    p.kpi_qa,
```

```
    p.kpi_avg_prod,
    CASE
        WHEN p.kpi_qa < 97 OR p.kpi_avg_prod < 98 THEN true
        ELSE false
    END as is_underperforming,
    CASE
        WHEN p.kpi_qa < 97 THEN 'QA'
        WHEN p.kpi_avg_prod < 98 THEN 'Production'
        ELSE NULL
    END as underperforming_metric
FROM performance_data p
WHERE p.kpi_qa < 97 OR p.kpi_avg_prod < 98;
```

2. v_agent_warning_status

Purpose: Current warning status for all agents

```
CREATE OR REPLACE VIEW v_agent_warning_status AS
SELECT
    a.agent_email,
    a.agent_name,
    a.warning_type,
    COUNT(*) FILTER (WHERE a.action_type = 'Verbal Warning') as verbal_count,
    COUNT(*) FILTER (WHERE a.action_type = 'Written Warning') as written_count,
    COUNT(*) FILTER (WHERE a.action_type = 'Final Warning') as final_count,
    MAX(a.expiration_date) as next_expiration,
    CASE
        WHEN COUNT(*) FILTER (WHERE a.action_type = 'Written Warning') >= 2 THEN
'CRITICAL'
        WHEN COUNT(*) FILTER (WHERE a.action_type = 'Written Warning') >= 1 THEN
'HIGH'
        WHEN COUNT(*) FILTER (WHERE a.action_type = 'Verbal Warning') >= 2 THEN
'MEDIUM'
        WHEN COUNT(*) FILTER (WHERE a.action_type = 'Verbal Warning') >= 1 THEN
'LOW'
        ELSE 'NORMAL'
    END as risk_level
FROM action_logs a
WHERE a.is_active = true
GROUP BY a.agent_email, a.agent_name, a.warning_type;
```

3. v_monthly_performance_summary

Purpose: Monthly aggregated performance metrics

```
CREATE OR REPLACE VIEW v_monthly_performance_summary AS
SELECT
    p.month_name,
    p.year_num,
```

```
    p.client,
    p.category,
    COUNT(DISTINCT p.agent_email) as total_agents,
    COUNT(DISTINCT CASE
        WHEN p.kpi_qa < 97 OR p.kpi_avg_prod < 98
        THEN p.agent_email
    END) as underperforming_agents,
    AVG(p.kpi_qa) as avg_qa_score,
    AVG(p.kpi_avg_prod) as avg_prod_score,
    COUNT(DISTINCT p.week_range) as total_weeks
FROM performance_data p
GROUP BY p.month_name, p.year_num, p.client, p.category;
```

Stored Procedures

1. sp_expire_warnings

Purpose: Automatically expire warnings past their expiration date

```
CREATE OR REPLACE FUNCTION sp_expire_warnings()
RETURNS INTEGER AS $$
DECLARE
    expired_count INTEGER;
BEGIN
    UPDATE action_logs
    SET is_active = false,
        updated_at = CURRENT_TIMESTAMP
    WHERE is_active = true
    AND expiration_date IS NOT NULL
    AND expiration_date < CURRENT_DATE;

    GET DIAGNOSTICS expired_count = ROW_COUNT;

    RETURN expired_count;
END;
$$ LANGUAGE plpgsql;

-- Schedule to run daily
-- (Use cron job or scheduler)
```

2. sp_calculate_monthly_summary

Purpose: Generate monthly summary aggregations

```
CREATE OR REPLACE FUNCTION sp_calculate_monthly_summary(
    p_month INTEGER,
    p_year INTEGER
)
```

```
RETURNS VOID AS $$
BEGIN
    DELETE FROM monthly_summaries
    WHERE month_num = p_month AND year_num = p_year;

    INSERT INTO monthly_summaries (
        month_num, year_num, client, category,
        total_aftes, underperformers, avg_score,
        avg_qa_score, avg_prod_score
    )
    SELECT
        p.month_num,
        p.year_num,
        p.client,
        p.category,
        COUNT(DISTINCT p.agent_email) as total_aftes,
        COUNT(DISTINCT CASE
            WHEN p.kpi_qa < 97 OR p.kpi_avg_prod < 98
            THEN p.agent_email
        END) as underperformers,
        (AVG(p.kpi_qa) + AVG(p.kpi_avg_prod)) / 2 as avg_score,
        AVG(p.kpi_qa) as avg_qa_score,
        AVG(p.kpi_avg_prod) as avg_prod_score
    FROM performance_data p
    WHERE p.month_num = p_month AND p.year_num = p_year
    GROUP BY p.month_num, p.year_num, p.client, p.category;
END;
$$ LANGUAGE plpgsql;
```

3. sp_get_agent_recommendations

Purpose: Get recommended actions for an agent

```
CREATE OR REPLACE FUNCTION sp_get_agent_recommendations(
    p_agent_email VARCHAR,
    p_category VARCHAR
)
RETURNS TABLE (
    case_type VARCHAR,
    recommended_action VARCHAR,
    priority INTEGER,
    is_critical BOOLEAN,
    notes TEXT
) AS $$
DECLARE
    v_verbal_count INTEGER;
    v_written_count INTEGER;
    v_underperforming_weeks INTEGER;
BEGIN
    -- Get warning counts
    SELECT
```

```
        COUNT(*) FILTER (WHERE action_type = 'Verbal Warning'),
        COUNT(*) FILTER (WHERE action_type = 'Written Warning')
    INTO v_verbal_count, v_written_count
    FROM action_logs
    WHERE agent_email = p_agent_email
    AND warning_type = p_category
    AND is_active = true;

    -- Get underperforming weeks in current month
    SELECT COUNT(*)
    INTO v_underperforming_weeks
    FROM performance_data
    WHERE agent_email = p_agent_email
    AND (kpi_qa < 97 OR kpi_avg_prod < 98)
    AND month_num = EXTRACT(MONTH FROM CURRENT_DATE)
    AND year_num = EXTRACT(YEAR FROM CURRENT_DATE);

    -- Apply business logic (Cases A-E)
    IF v_written_count >= 2 AND v_underperforming_weeks > 0 THEN
        -- Case C
        RETURN QUERY SELECT
            'C'::VARCHAR,
            'Prepare for Employee Termination'::VARCHAR,
            1::INTEGER,
            true::BOOLEAN,
            'Agent has 2 Written Warnings and continues to underperform. Consult
HR.'::TEXT;
    ELSIF v_verbal_count >= 2 AND v_written_count = 0 AND v_underperforming_weeks
> 0 THEN
        -- Case B
        RETURN QUERY SELECT
            'B'::VARCHAR,
            'Issue Written Warning'::VARCHAR,
            2::INTEGER,
            true::BOOLEAN,
            'Agent has 2 Verbal Warnings and continues to underperform. Escalate
to Written Warning.'::TEXT;
    ELSIF v_verbal_count = 1 AND v_underperforming_weeks > 0 THEN
        -- Case A
        RETURN QUERY SELECT
            'A'::VARCHAR,
            'Issue 2nd Verbal Warning + Coaching'::VARCHAR,
            3::INTEGER,
            false::BOOLEAN,
            'Agent has 1 Verbal Warning and new underperformance. Issue 2nd Verbal
+ coaching.'::TEXT;
    ELSE
        -- Monitor
        RETURN QUERY SELECT
            'MONITOR'::VARCHAR,
            'Continue Monitoring'::VARCHAR,
            5::INTEGER,
            false::BOOLEAN,
            'Agent performance within acceptable range.'::TEXT;
```

```
    END IF;
END;
$$ LANGUAGE plpgsql;
```

Data Dictionary

performance_data Fields

Field	Type	Null	Description
id	SERIAL	NO	Primary key, auto-incrementing
agent_email	VARCHAR(255)	NO	Agent's email address (unique identifier)
agent_id	VARCHAR(100)	YES	Optional employee ID
agent_name	VARCHAR(255)	YES	Agent's full name
position	VARCHAR(100)	YES	Job title/position
office	VARCHAR(100)	YES	Office location
client	VARCHAR(255)	NO	Client name
task	VARCHAR(255)	YES	Specific task type
category	VARCHAR(255)	NO	Performance category
kpi_qa	DECIMAL(5,2)	NO	QA performance score (0-200)
flag_qa	VARCHAR(50)	YES	QA flag (Critical, Low, Normal, Good, Great)
kpi_avg_prod	DECIMAL(5,2)	NO	Production performance score (0-200)
flag_prod	VARCHAR(50)	YES	Production flag
week_range	VARCHAR(50)	NO	Display string (e.g., "12/01 - 12/07")
start_date	DATE	NO	Week start date (Sunday)
end_date	DATE	NO	Week end date (Saturday)
month_num	INTEGER	NO	Month number (1-12)
month_name	VARCHAR(20)	NO	Month name (January-December)
year_num	INTEGER	NO	Year (4-digit)
created_at	TIMESTAMP	NO	Record creation timestamp
updated_at	TIMESTAMP	NO	Last update timestamp

action_logs Fields

Field	Type	Null	Description

Field	Type	Null	Description
id	SERIAL	NO	Primary key
agent_email	VARCHAR(255)	NO	Agent receiving the action
agent_name	VARCHAR(255)	YES	Agent's name
action_date	DATE	NO	Date action was taken
action_type	VARCHAR(100)	NO	Type of action (Coaching, Verbal Warning, etc.)
description	TEXT	NO	Detailed description (min 10 chars)
taken_by	VARCHAR(255)	NO	Manager/supervisor who took action
week_start_date	DATE	YES	Week start this action applies to
week_end_date	DATE	YES	Week end this action applies to
client	VARCHAR(255)	YES	Client name
category	VARCHAR(255)	YES	Task category
warning_type	VARCHAR(100)	YES	Warning category (Production, QA, Other)
expiration_date	DATE	YES	When warning expires (null for permanent)
is_active	BOOLEAN	NO	Whether warning is currently active
created_at	TIMESTAMP	NO	Record creation timestamp
updated_at	TIMESTAMP	NO	Last update timestamp

Sample Queries

1. Get All Underperforming Agents for Current Month

```sql
SELECT
    p.agent_email,
    p.agent_name,
    p.client,
    COUNT(*) as underperforming_weeks,
    AVG(p.kpi_qa) as avg_qa,
    AVG(p.kpi_avg_prod) as avg_prod
FROM performance_data p
WHERE p.month_num = EXTRACT(MONTH FROM CURRENT_DATE)
AND p.year_num = EXTRACT(YEAR FROM CURRENT_DATE)
AND (p.kpi_qa < 97 OR p.kpi_avg_prod < 98)
GROUP BY p.agent_email, p.agent_name, p.client
ORDER BY underperforming_weeks DESC;
```

2. Get Agent Warning History

```sql
SELECT
    a.action_date,
    a.action_type,
    a.warning_type,
    a.description,
    a.taken_by,
    a.expiration_date,
    a.is_active,
    CASE
        WHEN a.expiration_date < CURRENT_DATE THEN 'Expired'
        WHEN a.is_active = true THEN 'Active'
        ELSE 'Inactive'
    END as status
FROM action_logs a
WHERE a.agent_email = 'john.doe@onq.com'
ORDER BY a.action_date DESC;
```

3. Get Monthly Summary by Client

```sql
SELECT
    p.client,
    COUNT(DISTINCT p.agent_email) as total_agents,
    COUNT(DISTINCT CASE
        WHEN p.kpi_qa < 97 OR p.kpi_avg_prod < 98
        THEN p.agent_email
    END) as underperformers,
    ROUND(AVG(p.kpi_qa), 2) as avg_qa_score,
    ROUND(AVG(p.kpi_avg_prod), 2) as avg_prod_score
FROM performance_data p
WHERE p.month_name = 'December'
AND p.year_num = 2025
GROUP BY p.client
ORDER BY underperformers DESC;
```

4. Find Agents with Multiple Active Warnings

```sql
SELECT
    a.agent_email,
    a.agent_name,
    a.warning_type,
    COUNT(*) as warning_count,
    STRING_AGG(a.action_type, ', ') as warning_types,
    MIN(a.expiration_date) as earliest_expiration
FROM action_logs a
WHERE a.is_active = true
GROUP BY a.agent_email, a.agent_name, a.warning_type
HAVING COUNT(*) > 1
ORDER BY warning_count DESC;
```

5. Get Consecutive Underperforming Weeks

```
WITH numbered_weeks AS (
    SELECT
        agent_email,
        agent_name,
        week_range,
        start_date,
        CASE
            WHEN kpi_qa < 97 OR kpi_avg_prod < 98 THEN 1
            ELSE 0
        END as is_underperforming,
        ROW_NUMBER() OVER (PARTITION BY agent_email ORDER BY start_date) as
week_num
    FROM performance_data
    WHERE month_num = 12 AND year_num = 2025
),
grouped_weeks AS (
    SELECT
        agent_email,
        agent_name,
        is_underperforming,
        week_num - ROW_NUMBER() OVER (
            PARTITION BY agent_email, is_underperforming
            ORDER BY week_num
        ) as grp
    FROM numbered_weeks
)
SELECT
    agent_email,
    agent_name,
    COUNT(*) as consecutive_underperforming_weeks
FROM grouped_weeks
WHERE is_underperforming = 1
GROUP BY agent_email, agent_name, grp
HAVING COUNT(*) >= 3
ORDER BY consecutive_underperforming_weeks DESC;
```

6. Get At-Risk Agents (3+ Consecutive Weeks)

```
WITH weekly_performance AS (
    SELECT
        agent_email,
        agent_name,
        start_date,
        kpi_qa,
        kpi_avg_prod,
        CASE
```

```
            WHEN kpi_qa < 97 OR kpi_avg_prod < 98 THEN 1
            ELSE 0
        END as is_underperforming,
        LAG(CASE
            WHEN kpi_qa < 97 OR kpi_avg_prod < 98 THEN 1
            ELSE 0
        END, 1, 0) OVER (PARTITION BY agent_email ORDER BY start_date) as
prev_week_1,
        LAG(CASE
            WHEN kpi_qa < 97 OR kpi_avg_prod < 98 THEN 1
            ELSE 0
        END, 2, 0) OVER (PARTITION BY agent_email ORDER BY start_date) as
prev_week_2
    FROM performance_data
    WHERE month_num = EXTRACT(MONTH FROM CURRENT_DATE)
    AND year_num = EXTRACT(YEAR FROM CURRENT_DATE)
)
SELECT DISTINCT
    agent_email,
    agent_name,
    'HIGH' as risk_level,
    '3+ consecutive underperforming weeks' as reason
FROM weekly_performance
WHERE is_underperforming = 1
AND prev_week_1 = 1
AND prev_week_2 = 1;
```

Data Flow

Data Ingestion Flow

```
1. External System (HR/Payroll)
   ↓
2. ETL Process (Nightly)
   ↓
3. Staging Tables
   ↓
4. Data Validation
   ↓
5. performance_data table (INSERT/UPDATE)
   ↓
6. Triggers update monthly_summaries
   ↓
7. Frontend API queries data
```

Action Logging Flow

```
1. User submits action via frontend
   ↓
2. Backend API validates data
   ↓
3. Calculate expiration_date
   ↓
4. INSERT into action_logs table
   ↓
5. Trigger updates agent_warnings view
   ↓
6. Return success to frontend
```

Backup & Maintenance

Backup Strategy

Daily Backups:

```
pg_dump -U postgres -d qperform_db -F c -f qperform_backup_$(date +%Y%m%d).dump
```

Weekly Full Backups:

```
pg_basebackup -D /backup/weekly/$(date +%Y%m%d) -F tar -z -P
```

Retention Policy:

- Daily backups: 30 days
- Weekly backups: 90 days
- Monthly backups: 1 year

Maintenance Tasks

Daily:

- Expire warnings: `SELECT sp_expire_warnings();`
- Vacuum analyze: `VACUUM ANALYZE;`

Weekly:

- Reindex: `REINDEX DATABASE qperform_db;`
- Update statistics: `ANALYZE;`

Monthly:

- Generate monthly summaries: `SELECT sp_calculate_monthly_summary(month, year);`
- Archive old data (>2 years)

- Check table bloat and cleanup

Data Retention

- `performance_data`: 3 years
- `action_logs`: Permanent (legal requirement)
- `monthly_summaries`: Permanent
- `users`: Active users only (soft delete)

Version History

Version	Date	Changes
1.0	2025-12-09	Initial database structure documentation

Document Maintained By: Database Team **Contact:** dba@qsoftware.cloud **Next Review Date:** 2026-03-09