Enterprise-Grade SQL Code

```
-- SQL Sample: Enterprise-level Database System for a SaaS Platform
-- Purpose: Demonstrates mastery in database design, normalization, indexing, views, CTEs, window functions, transactions, and optimization

-- ========================================
-- SCHEMA DESIGN - NORMALIZED DATA MODEL
-- ========================================

-- USERS: Contains user profile information
CREATE TABLE users (
    user_id         SERIAL PRIMARY KEY,
    username        VARCHAR(50) NOT NULL UNIQUE,
    email           VARCHAR(100) NOT NULL UNIQUE,
    created_at      TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    is_active       BOOLEAN DEFAULT TRUE
);

-- ROLES: Role-based access control (RBAC)
CREATE TABLE roles (
    role_id         SERIAL PRIMARY KEY,
    role_name       VARCHAR(50) UNIQUE NOT NULL
);

-- USER_ROLES: Many-to-many relationship between users and roles
CREATE TABLE user_roles (
    user_id         INT REFERENCES users(user_id),
    role_id         INT REFERENCES roles(role_id),
    assigned_at     TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (user_id, role_id)
);

-- SUBSCRIPTIONS: Each user may have an active or expired subscription
CREATE TABLE subscriptions (
    subscription_id     SERIAL PRIMARY KEY,
    user_id             INT REFERENCES users(user_id),
    plan_name           VARCHAR(50) NOT NULL,
    price               NUMERIC(10, 2),
```

```
    start_date          DATE NOT NULL,
    end_date            DATE,
    is_active           BOOLEAN DEFAULT TRUE
);

-- AUDIT_LOG: Tracks user actions
CREATE TABLE audit_log (
    log_id          BIGSERIAL PRIMARY KEY,
    user_id         INT REFERENCES users(user_id),
    action_type     VARCHAR(100),
    details         TEXT,
    logged_at       TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- ========================================
-- INDEXING STRATEGY FOR PERFORMANCE
-- ========================================

CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_roles_role_name ON roles(role_name);
CREATE INDEX idx_audit_user_time ON audit_log(user_id, logged_at DESC);
CREATE INDEX idx_subscriptions_active ON subscriptions(user_id, is_active);

-- ========================================
-- VIEW: ACTIVE USERS WITH ACTIVE SUBSCRIPTIONS
-- ========================================

CREATE VIEW active_users_view AS
SELECT
    u.user_id,
    u.username,
    u.email,
    s.plan_name,
    s.price,
    s.start_date,
    s.end_date
FROM users u
JOIN subscriptions s ON u.user_id = s.user_id
WHERE u.is_active = TRUE AND s.is_active = TRUE;
```

```
-- ========================================
-- CTEs: Common Table Expressions & Analytics
-- ========================================

WITH monthly_registrations AS (
    SELECT
        DATE_TRUNC('month', created_at) AS month,
        COUNT(*) AS user_count
    FROM users
    WHERE created_at >= CURRENT_DATE - INTERVAL '12 months'
    GROUP BY 1
    ORDER BY 1
)
SELECT * FROM monthly_registrations;

-- ========================================
-- WINDOW FUNCTIONS: Subscription rank per user
-- ========================================

SELECT
    user_id,
    subscription_id,
    plan_name,
    start_date,
    end_date,
    RANK() OVER (PARTITION BY user_id ORDER BY start_date DESC) AS
subscription_rank
FROM subscriptions;

-- ========================================
-- TRIGGERS: Auto-audit for user actions
-- ========================================

CREATE OR REPLACE FUNCTION log_user_insert()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO audit_log(user_id, action_type, details)
    VALUES (NEW.user_id, 'CREATE_USER', 'New user created: ' || NEW.username);
    RETURN NEW;
END;
```

```
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_log_user_insert
AFTER INSERT ON users
FOR EACH ROW
EXECUTE FUNCTION log_user_insert();

-- ========================================
-- TRANSACTIONS & CONCURRENCY
-- ========================================

BEGIN;

INSERT INTO user_roles(user_id, role_id)
VALUES (1, 2);

INSERT INTO audit_log(user_id, action_type, details)
VALUES (1, 'ROLE_ASSIGNED', 'Role ID 2 assigned to user 1');

COMMIT;

-- ========================================
-- ADVANCED SQL: Detect inactive users (no login in 90 days)
-- ========================================

SELECT
   u.user_id,
   u.username,
   MAX(a.logged_at) AS last_login
FROM users u
LEFT JOIN audit_log a ON u.user_id = a.user_id AND a.action_type = 'LOGIN'
GROUP BY u.user_id, u.username
HAVING MAX(a.logged_at) < CURRENT_DATE - INTERVAL '90 days' OR
MAX(a.logged_at) IS NULL;

-- ========================================
-- UPSERT / MERGE: PostgreSQL 9.5+
-- ========================================

INSERT INTO users (username, email)
```

```
VALUES ('john_doe', 'john@example.com')
ON CONFLICT (username) DO UPDATE
SET email = EXCLUDED.email;

-- ========================================
-- PARTITIONING FOR BIG DATA HANDLING
-- ========================================

CREATE TABLE audit_log_partitioned (
   log_id          BIGSERIAL NOT NULL,
   user_id         INT NOT NULL,
   action_type     VARCHAR(100),
   details         TEXT,
   logged_at       TIMESTAMP NOT NULL,
   PRIMARY KEY (log_id, logged_at)
) PARTITION BY RANGE (logged_at);

CREATE TABLE audit_log_2025_01 PARTITION OF audit_log_partitioned
   FOR VALUES FROM ('2025-01-01') TO ('2025-02-01');

CREATE TABLE audit_log_2025_02 PARTITION OF audit_log_partitioned
   FOR VALUES FROM ('2025-02-01') TO ('2025-03-01');

-- ========================================
-- JSONB USAGE (Semi-Structured Data in PostgreSQL)
-- ========================================

ALTER TABLE users ADD COLUMN preferences JSONB DEFAULT '{}';

UPDATE users
SET preferences = jsonb_set(preferences, '{theme}', '"dark"', true)
WHERE user_id = 1;

SELECT * FROM users
WHERE preferences ->> 'theme' = 'dark';

-- ========================================
-- CLEANUP: Cascade deletes with foreign keys
-- ========================================
```

```
ALTER TABLE subscriptions
   DROP CONSTRAINT subscriptions_user_id_fkey,
   ADD CONSTRAINT subscriptions_user_id_fkey FOREIGN KEY (user_id)
   REFERENCES users(user_id) ON DELETE CASCADE;

-- ========================================
-- OPTIMIZATION HINTS
-- ========================================

EXPLAIN ANALYZE
SELECT * FROM active_users_view WHERE plan_name = 'Premium';

VACUUM ANALYZE;

CREATE MATERIALIZED VIEW monthly_summary AS
SELECT
   DATE_TRUNC('month', s.start_date) AS month,
   COUNT(*) AS subscriptions,
   SUM(s.price) AS revenue
FROM subscriptions s
GROUP BY 1;

REFRESH MATERIALIZED VIEW monthly_summary;
```