PostgreSQL Database Creation Script for Apartment Complex Account Management System

sql

```sql
-- ============================================
-- Database Creation and Configuration
-- ============================================

-- Create the database with appropriate settings
CREATE DATABASE apartment_accounting
    WITH
    OWNER = postgres
    ENCODING = 'UTF8'
    LC_COLLATE = 'en_US.UTF-8'
    LC_CTYPE = 'en_US.UTF-8'
    TABLESPACE = pg_default
    CONNECTION LIMIT = -1
    IS_TEMPLATE = False;

-- Connect to the new database
\c apartment_accounting

-- Create a dedicated schema for our application
CREATE SCHEMA accounting;
COMMENT ON SCHEMA accounting IS 'Schema for apartment complex account management syste
m';

-- ============================================
-- Role and Permission Configuration
-- ============================================

-- Create application roles with least privilege principle
CREATE ROLE accounting_admin WITH NOLOGIN;
COMMENT ON ROLE accounting_admin IS 'Administrative role with full access to accounting sche
ma';

CREATE ROLE accounting_app WITH LOGIN PASSWORD 'secure_password_123' NOINHERIT;
COMMENT ON ROLE accounting_app IS 'Application role for normal operations';

CREATE ROLE accounting_readonly WITH LOGIN PASSWORD 'readonly_password_456' NOINHE
RIT;
COMMENT ON ROLE accounting_readonly IS 'Read-only role for reporting';
```

```sql
-- Grant schema privileges
GRANT USAGE ON SCHEMA accounting TO accounting_admin;
GRANT USAGE ON SCHEMA accounting TO accounting_app;
GRANT USAGE ON SCHEMA accounting TO accounting_readonly;

-- Grant object creation to admin
GRANT CREATE ON SCHEMA accounting TO accounting_admin;

-- Set default privileges for future objects
ALTER DEFAULT PRIVILEGES IN SCHEMA accounting
    GRANT ALL PRIVILEGES ON TABLES TO accounting_admin;

ALTER DEFAULT PRIVILEGES IN SCHEMA accounting
    GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO accounting_app;

ALTER DEFAULT PRIVILEGES IN SCHEMA accounting
    GRANT SELECT ON TABLES TO accounting_readonly;

-- ========================================
-- Core Tables (as specified in LLD)
-- ========================================

-- Accounts table (as per LLD specification)
CREATE TABLE accounting.accounts (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(255) NOT NULL,
    type VARCHAR(50) NOT NULL CHECK (type IN ('INCOME', 'EXPENSE', 'ASSET', 'LIABILITY')),
    description TEXT,
    opening_balance DECIMAL(19,4) NOT NULL,
    current_balance DECIMAL(19,4) NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT_TIMESTAMP,
    created_by VARCHAR(100) NOT NULL DEFAULT current_user,
    updated_by VARCHAR(100) NOT NULL DEFAULT current_user
);

COMMENT ON TABLE accounting.accounts IS 'Stores all financial accounts for the apartment compl
ex';
COMMENT ON COLUMN accounting.accounts.type IS 'Account type: INCOME, EXPENSE, ASSET,
or LIABILITY';
COMMENT ON COLUMN accounting.accounts.opening_balance IS 'Initial balance when account wa
s created';
COMMENT ON COLUMN accounting.accounts.current_balance IS 'Current balance after all transacti
ons';

-- Transactions table (as per LLD specification with audit fields)
```

```sql
CREATE TABLE accounting.transactions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    account_id UUID NOT NULL REFERENCES accounting.accounts(id),
    contra_account_id UUID NOT NULL REFERENCES accounting.accounts(id),
    transaction_date DATE NOT NULL,
    amount DECIMAL(19,4) NOT NULL CHECK (amount > 0),
    description VARCHAR(500),
    reference_number VARCHAR(100),
    is_void BOOLEAN NOT NULL DEFAULT FALSE,
    created_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT_TIMESTAMP,
    created_by VARCHAR(100) NOT NULL DEFAULT current_user,
    CHECK (account_id != contra_account_id)
);

COMMENT ON TABLE accounting.transactions IS 'Records all financial transactions between accoun
ts';
COMMENT ON COLUMN accounting.transactions.contra_account_id IS 'The counterparty account f
or the transaction';
COMMENT ON COLUMN accounting.transactions.is_void IS 'Flag for voided/cancelled transactions'
;

-- Balance history table (as per LLD specification)
CREATE TABLE accounting.balance_history (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    account_id UUID NOT NULL REFERENCES accounting.accounts(id),
    balance_date DATE NOT NULL,
    balance DECIMAL(19,4) NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(account_id, balance_date)
);

COMMENT ON TABLE accounting.balance_history IS 'Historical record of account balances for repo
rting';
COMMENT ON COLUMN accounting.balance_history.balance_date IS 'Date when the balance was r
ecorded';

-- ============================================
-- Audit Trail Implementation
-- ============================================

-- Audit log table for tracking all DML changes
CREATE TABLE accounting.audit_log (
    id BIGSERIAL PRIMARY KEY,
    table_name VARCHAR(100) NOT NULL,
    record_id UUID NOT NULL,
    operation VARCHAR(10) NOT NULL CHECK (operation IN ('INSERT', 'UPDATE', 'DELETE')),
```

```
    old_values JSONB,
    new_values JSONB,
    changed_by VARCHAR(100) NOT NULL,
    changed_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT_TIMESTAMP
);

COMMENT ON TABLE accounting.audit_log IS 'Audit trail for all data modifications in the system';
COMMENT ON COLUMN accounting.audit_log.old_values IS 'JSON representation of the record bef
ore change';
COMMENT ON COLUMN accounting.audit_log.new_values IS 'JSON representation of the record af
ter change';

CREATE INDEX idx_audit_log_table_record ON accounting.audit_log(table_name, record_id);
CREATE INDEX idx_audit_log_changed_at ON accounting.audit_log(changed_at);

-- ================================================
-- Indexes (as specified in LLD plus recommended indexes)
-- ================================================

-- Indexes specified in LLD
CREATE INDEX idx_transactions_account_id ON accounting.transactions(account_id);
CREATE INDEX idx_transactions_contra_account_id ON accounting.transactions(contra_account_id)
;
CREATE INDEX idx_transactions_date ON accounting.transactions(transaction_date);
CREATE INDEX idx_transactions_void_status ON accounting.transactions(is_void) WHERE is_void
= TRUE;
CREATE INDEX idx_balance_history_account_date ON accounting.balance_history(account_id, bala
nce_date);

-- Additional recommended indexes
CREATE INDEX idx_accounts_type ON accounting.accounts(type);
CREATE INDEX idx_transactions_created_at ON accounting.transactions(created_at);
CREATE INDEX idx_accounts_name ON accounting.accounts(name);

-- ================================================
-- Reporting Views
-- ================================================

-- View for current account balances
CREATE VIEW accounting.current_balances AS
SELECT
    a.id,
    a.name,
    a.type,
    a.current_balance,
    a.updated_at as last_updated
```

```
FROM
    accounting.accounts a
ORDER BY
    a.type, a.name;

COMMENT ON VIEW accounting.current_balances IS 'Current balance summary for all accounts';

-- View for transaction history with account names
CREATE VIEW accounting.transaction_history AS
SELECT
    t.id,
    t.transaction_date,
    a1.name as account_name,
    a2.name as contra_account_name,
    t.amount,
    t.description,
    t.reference_number,
    t.is_void,
    t.created_at
FROM
    accounting.transactions t
JOIN
    accounting.accounts a1 ON t.account_id = a1.id
JOIN
    accounting.accounts a2 ON t.contra_account_id = a2.id
ORDER BY
    t.transaction_date DESC, t.created_at DESC;

COMMENT ON VIEW accounting.transaction_history IS 'Detailed transaction history with account n
ames';

-- View for monthly profit/loss (income minus expenses)
CREATE VIEW accounting.monthly_profit_loss AS
SELECT
    date_trunc('month', t.transaction_date) as month,
    SUM(CASE WHEN a.type = 'INCOME' THEN t.amount ELSE 0 END) as total_income,
    SUM(CASE WHEN a.type = 'EXPENSE' THEN t.amount ELSE 0 END) as total_expenses,
    SUM(CASE WHEN a.type = 'INCOME' THEN t.amount ELSE -t.amount END) as net_profit_loss
FROM
    accounting.transactions t
JOIN
    accounting.accounts a ON t.account_id = a.id
WHERE
    t.is_void = FALSE
    AND a.type IN ('INCOME', 'EXPENSE')
GROUP BY
```

```
    date_trunc('month', t.transaction_date)
ORDER BY
    month DESC;

COMMENT ON VIEW accounting.monthly_profit_loss IS 'Monthly profit/loss calculation (income mi
nus expenses)';

-- ============================================
-- Final Permission Configuration
-- ============================================

-- Grant specific permissions to application role
GRANT SELECT, INSERT, UPDATE ON ALL TABLES IN SCHEMA accounting TO accounting_app;
GRANT USAGE ON ALL SEQUENCES IN SCHEMA accounting TO accounting_app;
GRANT SELECT ON ALL TABLES IN SCHEMA accounting TO accounting_readonly;
GRANT SELECT ON ALL SEQUENCES IN SCHEMA accounting TO accounting_readonly;

-- Grant all permissions to admin role
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA accounting TO accounting_admin;
GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA accounting TO accounting_admin;

-- ============================================
-- Database Configuration for Production
-- ============================================

-- Set search path for convenience
ALTER ROLE accounting_app SET search_path = accounting, public;
ALTER ROLE accounting_readonly SET search_path = accounting, public;
ALTER ROLE accounting_admin SET search_path = accounting, public;

-- Enable pgcrypto for UUID generation if not already enabled
CREATE EXTENSION IF NOT EXISTS pgcrypto;
```

Script Overview

1. **Database and Schema Setup**:
    - Creates a dedicated database with UTF-8 encoding
    - Establishes an isolated schema for the application
2. **Security Enhancements**:
    - Three distinct roles with least privilege:
        - `accounting_admin` for full control
        - `accounting_app` for normal operations

        - `accounting_readonly` for reporting
    - Secure password authentication for application roles
    - Schema isolation to prevent accidental access to other schemas
3. **Core Tables**:
    - `accounts` table exactly as specified in LLD
    - `transactions` table with all constraints from LLD
    - `balance_history` table for historical balance tracking
4. **Audit Trail**:
    - Comprehensive `audit_log` table tracking all DML changes
    - Records who changed what and when, with before/after values
    - No triggers needed (application will handle logging)
5. **Indexes**:
    - All indexes specified in the LLD
    - Additional recommended indexes for performance
6. **Reporting Views**:
    - `current_balances` for account summary
    - `transaction_history` with account names
    - `monthly_profit_loss` for financial reporting
7. **Production Configuration**:
    - Proper permissions for each role
    - Search path configuration
    - Extension setup for UUID generation

The script follows a logical order from database creation to final configuration, making it suitable for deployment in production environments.