PostgreSQL Database Requirements for Finance Flow

Overview

This document outlines the database requirements for migrating Finance Flow from Base44 cloud storage to a local PostgreSQL database. The application currently has 10 main entities and all UI features implemented, requiring a comprehensive database schema that maintains existing functionality.

Database Architecture

Connection & Configuration

- **Database**: PostgreSQL 14+ recommended
- **Connection Pool**: Recommended for production use
- **ORM**: Consider using Prisma, TypeORM, or native SQL queries
- **Migration Tool**: Required for schema versioning

Security Requirements

- **Authentication**: User-based data isolation
- **Encryption**: Sensitive data (passwords, account numbers) must be encrypted at rest
- **Access Control**: Row-level security for multi-user support (future)
- **Backup**: Automated backup strategy for financial data protection

Database Schema Design

1. Users Table

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    name VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    is_active BOOLEAN DEFAULT true
);
```

2. Income Table

Purpose: Track all income sources with categorization and frequency settings

```
CREATE TABLE income (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
    source VARCHAR(255) NOT NULL,
```

```
    amount DECIMAL(12,2) NOT NULL,
    frequency VARCHAR(50) NOT NULL, -- weekly, bi-weekly, monthly,
quarterly, yearly
    category VARCHAR(100) NOT NULL, -- salary, freelance, investment,
rental, business, other
    date_received DATE NOT NULL,
    description TEXT,
    is_recurring BOOLEAN DEFAULT false,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_income_user_id ON income(user_id);
CREATE INDEX idx_income_date ON income(date_received);
CREATE INDEX idx_income_category ON income(category);
```

3. Expenses Table

Purpose: Record all expenses with categories, payment methods, and recurring options

```
CREATE TABLE expenses (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
    description VARCHAR(255) NOT NULL,
    amount DECIMAL(12,2) NOT NULL,
    category VARCHAR(100) NOT NULL, -- housing, transportation, food,
utilities, healthcare, entertainment, shopping, insurance, debt_payments,
other
    date DATE NOT NULL,
    payment_method VARCHAR(50) NOT NULL, -- cash, credit_card, debit_card,
bank_transfer, check
    is_recurring BOOLEAN DEFAULT false,
    frequency VARCHAR(50), -- weekly, bi-weekly, monthly, quarterly,
yearly
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_expenses_user_id ON expenses(user_id);
CREATE INDEX idx_expenses_date ON expenses(date);
CREATE INDEX idx_expenses_category ON expenses(category);
```

4. Bills Table

Purpose: Manage recurring bills with due dates, payment status, and auto-pay settings

```
CREATE TABLE bills (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
```

```
    name VARCHAR(255) NOT NULL,
    amount DECIMAL(12,2) NOT NULL,
    due_date DATE NOT NULL,
    category VARCHAR(100) NOT NULL, -- utilities, rent, mortgage,
insurance, phone, internet, streaming, etc.
    status VARCHAR(50) DEFAULT 'pending', -- pending, paid, overdue
    is_recurring BOOLEAN DEFAULT true,
    frequency VARCHAR(50) DEFAULT 'monthly', -- weekly, bi-weekly,
monthly, quarterly, yearly
    auto_pay BOOLEAN DEFAULT false,
    notes TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_bills_user_id ON bills(user_id);
CREATE INDEX idx_bills_due_date ON bills(due_date);
CREATE INDEX idx_bills_status ON bills(status);
```

5. Debts Table

Purpose: Track debt accounts with balances, interest rates, and payment information

```
CREATE TABLE debts (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
    name VARCHAR(255) NOT NULL,
    balance DECIMAL(12,2) NOT NULL,
    original_amount DECIMAL(12,2),
    interest_rate DECIMAL(5,2) NOT NULL, -- APR percentage
    minimum_payment DECIMAL(12,2) NOT NULL,
    due_date DATE,
    type VARCHAR(100) NOT NULL, -- credit_card, personal_loan,
student_loan, mortgage, auto_loan, other
    priority VARCHAR(20) DEFAULT 'medium', -- high, medium, low
    notes TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_debts_user_id ON debts(user_id);
CREATE INDEX idx_debts_type ON debts(type);
CREATE INDEX idx_debts_priority ON debts(priority);
```

6. Assets Table

Purpose: Portfolio of assets with current values, purchase prices, and appreciation tracking

```
CREATE TABLE assets (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
    name VARCHAR(255) NOT NULL,
    value DECIMAL(15,2) NOT NULL,
    purchase_price DECIMAL(15,2),
    purchase_date DATE,
    category VARCHAR(100) NOT NULL, -- real_estate, vehicle, investment,
savings, retirement, other
    description TEXT,
    location VARCHAR(255),
    appreciation_rate DECIMAL(5,2), -- Annual appreciation percentage
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_assets_user_id ON assets(user_id);
CREATE INDEX idx_assets_category ON assets(category);
```

7. Credit Cards Table

Purpose: Credit card accounts with limits, balances, APR, and rewards information

```
CREATE TABLE credit_cards (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
    name VARCHAR(255) NOT NULL,
    credit_limit DECIMAL(12,2) NOT NULL,
    current_balance DECIMAL(12,2) NOT NULL DEFAULT 0,
    apr DECIMAL(5,2) NOT NULL,
    annual_fee DECIMAL(8,2) DEFAULT 0,
    rewards_program VARCHAR(255),
    payment_due_date DATE,
    minimum_payment DECIMAL(12,2),
    card_type VARCHAR(50), -- visa, mastercard, amex, discover, other
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_credit_cards_user_id ON credit_cards(user_id);
CREATE INDEX idx_credit_cards_active ON credit_cards(is_active);
```

8. Insurance Policies Table

Purpose: Insurance policies with coverage details, premiums, and policy information

```
CREATE TABLE insurance_policies (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
    policy_name VARCHAR(255) NOT NULL,
    provider VARCHAR(255) NOT NULL,
    policy_number VARCHAR(100),
    type VARCHAR(100) NOT NULL, -- health, auto, home, life, disability,
travel, pet
    premium DECIMAL(12,2) NOT NULL,
    premium_frequency VARCHAR(50) NOT NULL, -- monthly, quarterly, semi-
annual, annual
    coverage_amount DECIMAL(15,2),
    deductible DECIMAL(12,2),
    start_date DATE NOT NULL,
    end_date DATE,
    is_active BOOLEAN DEFAULT true,
    notes TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_insurance_policies_user_id ON
insurance_policies(user_id);
CREATE INDEX idx_insurance_policies_type ON insurance_policies(type);
CREATE INDEX idx_insurance_policies_active ON
insurance_policies(is_active);
```

9. Budget Items Table

Purpose: Recurring budget items for income and expenses with flexible frequency options

```
CREATE TABLE budget_items (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
    budget_id INTEGER REFERENCES budgets(id) ON DELETE CASCADE,
    name VARCHAR(255) NOT NULL,
    type VARCHAR(50) NOT NULL, -- income, expense
    amount DECIMAL(12,2) NOT NULL,
    category VARCHAR(100) NOT NULL,
    frequency VARCHAR(50) NOT NULL, -- weekly, bi-weekly, semi-monthly,
monthly, yearly
    start_date DATE NOT NULL,
    day_of_month_1 INTEGER, -- For bi-monthly budgets (1st payment day)
    day_of_month_2 INTEGER, -- For bi-monthly budgets (2nd payment day)
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_budget_items_user_id ON budget_items(user_id);
CREATE INDEX idx_budget_items_budget_id ON budget_items(budget_id);
CREATE INDEX idx_budget_items_type ON budget_items(type);
```

10. Budgets Table

Purpose: Budget periods with start and end dates for financial planning

```
CREATE TABLE budgets (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
    name VARCHAR(255) NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    notes TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_budgets_user_id ON budgets(user_id);
CREATE INDEX idx_budgets_date_range ON budgets(start_date, end_date);
```

11. Accounts Table

Purpose: Secure storage for financial account credentials and contact information

```
CREATE TABLE accounts (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
    account_name VARCHAR(255) NOT NULL,
    account_type VARCHAR(100) NOT NULL, -- bank, credit_card, investment,
insurance, utility, etc.
    website_url VARCHAR(500),
    username VARCHAR(255),
    password_encrypted TEXT, -- Encrypted password storage
    account_number_encrypted TEXT, -- Encrypted account number
    email_address VARCHAR(255),
    phone_number VARCHAR(50),
    notes TEXT,
    is_active BOOLEAN DEFAULT true,
    last_login DATE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_accounts_user_id ON accounts(user_id);
CREATE INDEX idx_accounts_type ON accounts(account_type);
CREATE INDEX idx_accounts_active ON accounts(is_active);
```

Data Migration Requirements

Migration Strategy

1. **Export Data**: Extract all data from Base44 entities
2. **Transform Data**: Convert Base44 format to PostgreSQL schema
3. **Load Data**: Import data with proper user associations
4. **Validate Data**: Ensure data integrity and completeness

Migration Scripts Needed

- User creation and authentication setup
- Entity data export from Base44
- Data transformation and validation
- PostgreSQL data import
- Foreign key and constraint validation
- Index creation for performance

API Layer Changes

Required Modifications

1. **Database Connection**: Replace Base44 client with PostgreSQL connection
2. **CRUD Operations**: Replace Base44 SDK calls with SQL queries
3. **Authentication**: Implement JWT or session-based auth
4. **Error Handling**: Database-specific error handling
5. **Validation**: Server-side data validation
6. **Transactions**: Multi-table operations with proper transactions

Example API Transformation

```
// Current Base44 approach
const income = await Income.list({ orderBy: [{ date_received: 'desc' }]
});

// New PostgreSQL approach
const income = await db.query(`
    SELECT * FROM income
    WHERE user_id = $1
    ORDER BY date_received DESC
`, [userId]);
```

Performance Considerations

Indexing Strategy

- **Primary Keys**: All tables have efficient primary keys
- **Foreign Keys**: Proper indexing on user_id and related foreign keys
- **Query Optimization**: Indexes on commonly filtered columns (date, category, status)
- **Composite Indexes**: For complex queries involving multiple columns

Optimization Recommendations

- **Connection Pooling**: Use pgBouncer or similar for connection management
- **Query Optimization**: Use EXPLAIN ANALYZE for slow queries
- **Caching**: Consider Redis for frequently accessed data
- **Partitioning**: For large datasets, consider table partitioning by date

Security Implementation

Data Encryption

- **At Rest**: Encrypt sensitive fields (passwords, account numbers)
- **In Transit**: Use SSL/TLS for all database connections
- **Application Level**: Encrypt sensitive data before storing

Access Control

- **User Isolation**: Row-level security to ensure users only access their data
- **API Security**: JWT tokens or session-based authentication
- **Database Users**: Separate database users for different access levels

Testing Requirements

Database Testing

- **Schema Validation**: Ensure all constraints and relationships work
- **Data Integrity**: Test foreign key constraints and cascading deletes
- **Performance Testing**: Load testing with realistic data volumes
- **Migration Testing**: Validate data migration accuracy

Integration Testing

- **API Endpoints**: Test all CRUD operations
- **Error Handling**: Database connection failures and constraint violations
- **Transaction Testing**: Multi-table operations and rollbacks

Deployment Considerations

Environment Setup

- **Development**: Local PostgreSQL with test data
- **Testing**: Separate test database with automated migrations
- **Production**: Optimized PostgreSQL configuration with backups

Backup Strategy

- **Regular Backups**: Daily automated backups
- **Point-in-Time Recovery**: Transaction log backups
- **Disaster Recovery**: Off-site backup storage

Monitoring & Maintenance

Database Monitoring

- **Performance Metrics**: Query performance and resource usage
- **Storage Monitoring**: Disk usage and growth patterns
- **Error Monitoring**: Failed queries and connection issues

Maintenance Tasks

- **Regular VACUUM**: Keep database optimized
- **Index Maintenance**: Monitor and optimize indexes
- **Statistics Updates**: Keep query planner statistics current

This database design maintains all existing functionality while providing a solid foundation for local PostgreSQL deployment with proper security, performance, and scalability considerations.