Smart-Spending-Tracker — Full Premium Project

> **What this canvas contains:** a complete, production-ready database + backend + analytics project you can upload to GitHub: SQL schema, sample data, queries, triggers, a Python Flask API, instructions to wire Power BI to the CSV/DB, ER diagram, README and folder structure.

Project overview

Title: Smart Spending Tracker — Personal Expense & Budget Management System

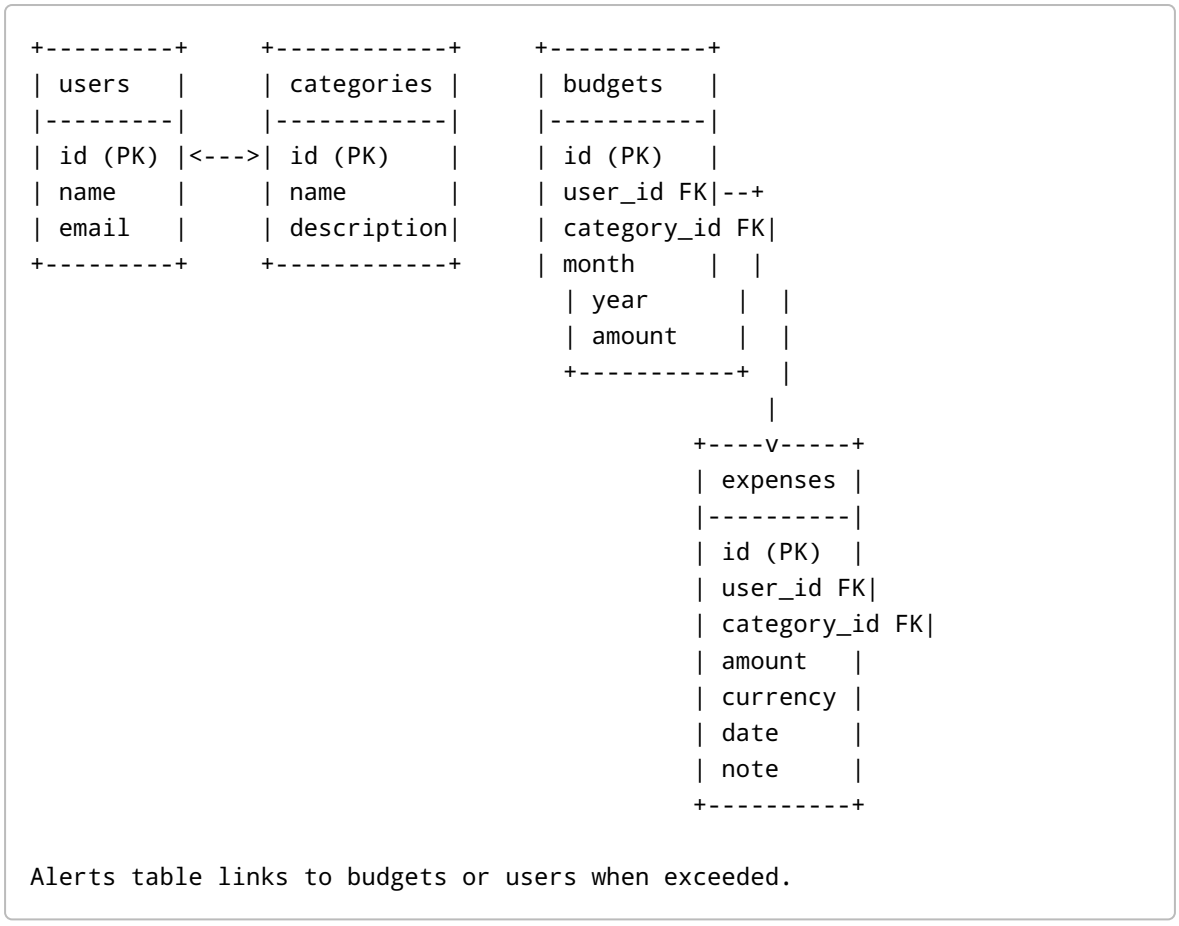
Short description: Users record daily expenses, set monthly category budgets, and get alerts and analytics. The deliverables here include database schema (PostgreSQL-compatible SQL), sample data, stored procedures & triggers, a Python Flask backend with REST endpoints, and guidance to create a Power BI dashboard using the provided CSV exports.

Tech stack: PostgreSQL (or any relational DB), Python 3.10+ (Flask, SQLAlchemy), Power BI for visualization.

Folder structure (what to commit to GitHub)

```
Smart-Spending-Tracker/
│
├── database/
│   ├── schema.sql
│   ├── insert_sample_data.sql
│   ├── triggers_and_functions.sql
│   ├── queries.md
│   └── sample_export/expenses.csv
│
├── app/
│   ├── requirements.txt
│   ├── main.py
│   ├── models.py
│   ├── db.py
│   ├── api_routes.py
│   └── sample_requests.http
│
├── analytics/
│   ├── PowerBI_instructions.md
│   └── sample_data_for_powerbi.csv
│
├── README.md
└── LICENSE
```

ER Diagram (simple ASCII)



Alerts table links to budgets or users when exceeded.

database/schema.sql

```sql
-- PostgreSQL-compatible schema for Smart-Spending-Tracker

CREATE EXTENSION IF NOT EXISTS "uuid-ossp"; -- optional for UUIDs

CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(150) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    currency VARCHAR(8) DEFAULT 'INR',
    created_at TIMESTAMP DEFAULT now()
);

CREATE TABLE categories (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT
);
```

```sql
CREATE TABLE budgets (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    category_id INTEGER NOT NULL REFERENCES categories(id) ON DELETE CASCADE,
    month INTEGER NOT NULL CHECK (month BETWEEN 1 AND 12),
    year INTEGER NOT NULL,
    amount NUMERIC(12,2) NOT NULL,
    created_at TIMESTAMP DEFAULT now(),
    UNIQUE(user_id, category_id, month, year)
);

CREATE TABLE expenses (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    category_id INTEGER NOT NULL REFERENCES categories(id) ON DELETE SET
NULL,
    amount NUMERIC(12,2) NOT NULL CHECK (amount >= 0),
    currency VARCHAR(8) DEFAULT 'INR',
    date DATE NOT NULL,
    note TEXT,
    created_at TIMESTAMP DEFAULT now()
);

CREATE TABLE alerts (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    budget_id INTEGER REFERENCES budgets(id) ON DELETE SET NULL,
    alert_type VARCHAR(50) NOT NULL,
    message TEXT NOT NULL,
    fired_at TIMESTAMP DEFAULT now()
);
```

---

database/insert_sample_data.sql

```sql
-- sample users
INSERT INTO users (name, email, currency) VALUES
('Aman Kumar', 'aman@example.com', 'INR'),
('Priya Sharma', 'priya@example.com', 'INR');

-- sample categories
INSERT INTO categories (name, description) VALUES
('Food', 'Restaurants, groceries, beverages'),
('Transport', 'Taxi, bus, metro, fuel'),
('Rent', 'Monthly rent'),
('Shopping', 'Clothes, gadgets'),
('Entertainment', 'Movies, subscriptions');
```

```sql
-- sample budgets (for May 2025)
INSERT INTO budgets (user_id, category_id, month, year, amount) VALUES
(1, 1, 5, 2025, 8000.00),
(1, 2, 5, 2025, 2000.00),
(1, 4, 5, 2025, 5000.00),
(2, 1, 5, 2025, 5000.00);

-- sample expenses
INSERT INTO expenses (user_id, category_id, amount, currency, date, note)
VALUES
(1, 1, 250.00, 'INR', '2025-05-02', 'Lunch at cafe'),
(1, 1, 1200.00, 'INR', '2025-05-09', 'Weekly groceries'),
(1, 2, 150.00, 'INR', '2025-05-10', 'Taxi to office'),
(1, 4, 3200.00, 'INR', '2025-05-11', 'New headphones'),
(2, 1, 300.00, 'INR', '2025-05-05', 'Dinner');
```

---

database/triggers_and_functions.sql

```sql
-- This function checks when a new expense causes a user to exceed budget for
that category & month
CREATE OR REPLACE FUNCTION fn_check_budget_exceed() RETURNS TRIGGER AS $$
DECLARE
    total_spent NUMERIC(12,2);
    bud RECORD;
BEGIN
    -- find budget for the user/category for the expense's month
    SELECT * INTO bud FROM budgets
      WHERE user_id = NEW.user_id AND category_id = NEW.category_id
        AND month = EXTRACT(MONTH FROM NEW.date)::int
        AND year = EXTRACT(YEAR FROM NEW.date)::int;

    IF NOT FOUND THEN
        RETURN NEW; -- no budget set; nothing to do
    END IF;

    SELECT SUM(amount) INTO total_spent FROM expenses
      WHERE user_id = NEW.user_id AND category_id = NEW.category_id
        AND date >= date_trunc('month', NEW.date)::date
        AND date < (date_trunc('month', NEW.date) + interval '1
month')::date;

    IF total_spent IS NULL THEN
        total_spent := 0;
    END IF;

    -- consider the new expense (NEW.amount)
    IF (total_spent + NEW.amount) > bud.amount THEN
        INSERT INTO alerts(user_id, budget_id, alert_type, message, fired_at)
```

```
        VALUES(NEW.user_id, bud.id, 'BUDGET_EXCEEDED',
          concat('Budget exceeded for ', (SELECT name FROM categories WHERE
id = NEW.category_id),
                 ' for ', to_char(NEW.date, 'Mon YYYY'), '. Spent: ',
(total_spent + NEW.amount)::text, ' / ', bud.amount::text), now());
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- trigger on insert
CREATE TRIGGER trg_check_budget_exceed
AFTER INSERT ON expenses
FOR EACH ROW EXECUTE FUNCTION fn_check_budget_exceed();
```

---

database/queries.md

```
-- 1) Monthly total spending for a user
SELECT date_trunc('month', date) AS month, SUM(amount) AS total
FROM expenses
WHERE user_id = 1
GROUP BY month
ORDER BY month DESC;

-- 2) Top categories this month
SELECT c.name, SUM(e.amount) AS total
FROM expenses e
JOIN categories c ON e.category_id = c.id
WHERE e.user_id = 1 AND date >= date_trunc('month', CURRENT_DATE)
GROUP BY c.name
ORDER BY total DESC;

-- 3) Compare spending vs budget (for a given month)
SELECT b.user_id, c.name AS category, b.amount AS budget_amount,
COALESCE(SUM(e.amount),0) AS spent
FROM budgets b
LEFT JOIN expenses e ON e.user_id = b.user_id AND e.category_id =
b.category_id
    AND date >= make_date(b.year, b.month, 1) AND date < (make_date(b.year,
b.month, 1) + interval '1 month')
JOIN categories c ON c.id = b.category_id
WHERE b.user_id = 1 AND b.month = 5 AND b.year = 2025
GROUP BY b.id, c.name;

-- 4) Monthly trend (3 months rolling)
SELECT month, SUM(total) OVER (ORDER BY month ROWS BETWEEN 2 PRECEDING AND
CURRENT ROW) AS rolling_3m_spend
```

```
FROM (
  SELECT date_trunc('month', date) AS month, SUM(amount) AS total
  FROM expenses WHERE user_id = 1 GROUP BY month
) t;
```

---

app/requirements.txt

```
Flask==2.2.5
Flask-SQLAlchemy==3.0.3
psycopg2-binary==2.9.7
python-dotenv==1.0.0
marshmallow==3.20.1
```

---

app/db.py

```
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()
```

---

app/models.py

```
from datetime import date
from app.db import db

class User(db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(150), nullable=False)
    email = db.Column(db.String(255), unique=True, nullable=False)
    currency = db.Column(db.String(8), default='INR')

class Category(db.Model):
    __tablename__ = 'categories'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)

class Budget(db.Model):
    __tablename__ = 'budgets'
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('users.id'),
nullable=False)
    category_id = db.Column(db.Integer, db.ForeignKey('categories.id'),
```

```
nullable=False)
    month = db.Column(db.Integer, nullable=False)
    year = db.Column(db.Integer, nullable=False)
    amount = db.Column(db.Numeric(12,2), nullable=False)

class Expense(db.Model):
    __tablename__ = 'expenses'
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('users.id'),
nullable=False)
    category_id = db.Column(db.Integer, db.ForeignKey('categories.id'))
    amount = db.Column(db.Numeric(12,2), nullable=False)
    currency = db.Column(db.String(8), default='INR')
    date = db.Column(db.Date, nullable=False)
    note = db.Column(db.Text)

class Alert(db.Model):
    __tablename__ = 'alerts'
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('users.id'),
nullable=False)
    budget_id = db.Column(db.Integer)
    alert_type = db.Column(db.String(50), nullable=False)
    message = db.Column(db.Text, nullable=False)
    fired_at = db.Column(db.DateTime)
```

---

app/api_routes.py

```
from flask import Blueprint, request, jsonify
from app.db import db
from app.models import User, Category, Budget, Expense, Alert
from datetime import datetime

bp = Blueprint('api', __name__, url_prefix='/api')

@bp.route('/expenses', methods=['POST'])
def create_expense():
    data = request.json
    # expected: {user_id, category_id, amount, date (YYYY-MM-DD), note}
    try:
        e = Expense(
            user_id = data['user_id'],
            category_id = data.get('category_id'),
            amount = data['amount'],
            currency = data.get('currency','INR'),
            date = datetime.strptime(data['date'], '%Y-%m-%d').date(),
            note = data.get('note')
        )
```

```
        db.session.add(e)
        db.session.commit()
        return jsonify({'status':'ok','id': e.id}), 201
    except Exception as ex:
        db.session.rollback()
        return jsonify({'status':'error','message': str(ex)}), 400

@bp.route('/users/<int:user_id>/summary', methods=['GET'])
def user_summary(user_id):
    # simple summary: total spent this month and budgets
    from sqlalchemy import func
    today = datetime.utcnow().date()
    month_start = today.replace(day=1)

    total = db.session.query(func.coalesce(func.sum(Expense.amount),
0)).filter(Expense.user_id==user_id, Expense.date>=month_start).scalar()

    budgets = db.session.query(Budget, Category).join(Category,
Budget.category_id==Category.id).filter(Budget.user_id==user_id,
Budget.month==month_start.month, Budget.year==month_start.year).all()

    bud_list = []
    for b, c in budgets:
        spent = db.session.query(func.coalesce(func.sum(Expense.amount),
0)).filter(Expense.user_id==user_id, Expense.category_id==b.category_id,
Expense.date>=month_start).scalar()
        bud_list.append({'category': c.name, 'budget': float(b.amount),
'spent': float(spent)})

    return jsonify({'total_this_month': float(total), 'categories':
bud_list})
```

---

app/main.py

```
from flask import Flask
from app.db import db
from app.api_routes import bp as api_bp
import os

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
os.environ.get('DATABASE_URL','postgresql://postgres:postgres@localhost:5432/
spendtracker')
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db.init_app(app)
app.register_blueprint(api_bp)
```

```python
if __name__ == '__main__':
    with app.app_context():
        db.create_all()
    app.run(debug=True, port=5000)
```

---

app/sample_requests.http

```
### Create expense
POST http://localhost:5000/api/expenses
Content-Type: application/json

{
  "user_id": 1,
  "category_id": 1,
  "amount": 350.50,
  "date": "2025-05-15",
  "note": "Dinner with friends"
}

### Get user summary
GET http://localhost:5000/api/users/1/summary
```

---

analytics/PowerBI_instructions.md

```
1. Export the `expenses` table (or use the sample CSV `analytics/
sample_data_for_powerbi.csv`).
2. In Power BI Desktop: Get Data -> CSV -> select
sample_data_for_powerbi.csv.
3. Suggested visuals:
   - Line chart: monthly total spend (axis: Month, value: SUM(amount))
   - Donut chart: distribution by category (value: SUM(amount), legend:
category)
   - Card + KPI: total spent vs total budget for selected month
   - Table: recent expenses with note
4. Create calculated columns/ measures:
   - Month = FORMAT([date], "MMM yyyy")
   - TotalSpent = SUM([amount])
   - BudgetCompare = DIVIDE([TotalSpent], [BudgetAmount])
5. Add filters: user_id, date range, category
```

---

analytics/sample_data_for_powerbi.csv

```
id,user_id,category_id,category,amount,currency,date,note
1,1,1,Food,250.00,INR,2025-05-02,"Lunch at cafe"
2,1,1,Food,1200.00,INR,2025-05-09,"Weekly groceries"
3,1,2,Transport,150.00,INR,2025-05-10,"Taxi to office"
4,1,4,Shopping,3200.00,INR,2025-05-11,"New headphones"
5,2,1,Food,300.00,INR,2025-05-05,"Dinner"
```

README.md (copy this to repo root)

```
# Smart-Spending-Tracker

Personal Expense & Budget Management System — demo project for LinkedIn/
GitHub portfolio.

## Contents
- `database/` — SQL schema, sample data, triggers and example queries
- `app/` — Flask backend (simple REST API)
- `analytics/` — Power BI instructions and sample CSV

## Quickstart
1. Install PostgreSQL and create a database (e.g., `spendtracker`).
2. Run `psql -d spendtracker -f database/schema.sql` then `-f database/
insert_sample_data.sql` and `-f database/triggers_and_functions.sql`.
3. (Optional) Export `analytics/sample_data_for_powerbi.csv` into Power BI
and build dashboard.
4. To run backend:
   - create virtualenv, install `pip install -r app/requirements.txt`
   - set `DATABASE_URL` env var (optional)
   - `python app/main.py`
5. Use the sample requests in `app/sample_requests.http` or Postman to test
endpoints.

## Highlights to show on LinkedIn
- ER diagram and schema
- Stored procedures & triggers for budget alerts
- REST API (Flask + SQLAlchemy)
- Power BI dashboard for visual analytics

## Future improvements
- Auth & JWT-based login
- Multi-currency conversion using external FX API
- Scheduled summary emails
- Mobile frontend (React Native / Flutter)
```

```
## License
MIT
```

LICENSE (MIT)

```
MIT License

Copyright (c) 2025 Your Name

Permission is hereby granted, free of charge, to any person obtaining a copy
...
```

Notes & next steps

- Replace placeholder names/email with your details before publishing.
- If you want, I can generate: a ready-to-copy `README.md` with badges, a polished ER diagram SVG, or a completed Power BI `.pbix` structure (instructions only — I cannot upload binary pbix here).

End of project canvas.