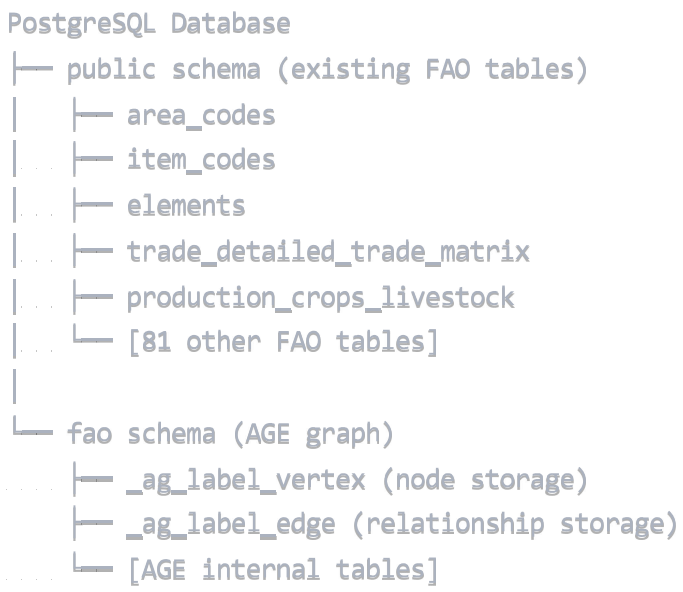
FAO Apache AGE Graph Database Implementation Guide

Overview

This guide documents the implementation of a graph database layer for the FAO database using Apache AGE (A Graph Extension). AGE enables graph database capabilities within PostgreSQL, allowing us to leverage graph algorithms and traversal queries while maintaining our existing relational data structure.

Architecture

System Design



Technology Stack

- **PostgreSQL 15+**: Primary database
- **Apache AGE**: Graph extension for PostgreSQL
- **Docker**: Containerized AGE deployment
- **Foreign Data Wrapper**: Connection between AGE and existing FAO database

Setup Instructions

1. Docker Deployment

yaml

```yaml
# docker-compose.yml
version: '3.8'

services:
  postgres-age:
    image: apache/age:latest
    container_name: fao-age
    environment:
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: fao_graph
    ports:
      - "5433:5432"
    volumes:
      - age_data:/var/lib/postgresql/data
    extra_hosts:
      - "host.docker.internal:host-gateway"

volumes:
  age_data:
```

2. Database Configuration

sql

```sql
-- Connect to AGE PostgreSQL
-- Enable required extensions
CREATE EXTENSION IF NOT EXISTS postgres_fdw;
CREATE EXTENSION IF NOT EXISTS age;
LOAD 'age';
SET search_path = ag_catalog, "$user", public;

-- Create foreign server connection to FAO database
CREATE SERVER fao_server
  FOREIGN DATA WRAPPER postgres_fdw
  OPTIONS (host 'host.docker.internal', port '5432', dbname 'fao');

-- User mapping for authentication
CREATE USER MAPPING FOR postgres
  SERVER fao_server
  OPTIONS (user 'postgres', password '${FAO_DB_PASSWORD}');

-- Import FAO schema
IMPORT FOREIGN SCHEMA public
  FROM SERVER fao_server
  INTO public;
```

3. Graph Creation

sql

```sql
-- Create the FAO graph
SELECT create_graph('fao');
```

Data Model

Node Types

Country Nodes (from area_codes)

sql

```
CREATE (c:Country {
    sql_id: Integer,       -- PostgreSQL primary key
    area_code: String,     -- FAO area code
    name: String,          -- Country/area name
    m49_code: String       -- UN M49 code
})
```

Item Nodes (from item_codes)

sql

```
CREATE (i:Item {
    sql_id: Integer,
    item_code: String,
    name: String,
    cpc_code: String,
    fbs_code: String,
    sdg_code: String
})
```

Element Nodes (from elements)

sql

```
CREATE (e:Element {
    sql_id: Integer,
    element_code: String,
    name: String
})
```

Relationship Types

TRADES_WITH (from trade_detailed_trade_matrix)

sql

```
(:Country)-[t:TRADES_WITH {
    item_code_id: Integer,
    element_code_id: Integer,
    year: Integer,
    value: Float,
    unit: String,
    flag_id: Integer
}]->(:Country)
```

PRODUCES (from production_crops_livestock)

sql

```
(:Country)-[p:PRODUCES {
    item_code_id: Integer,
    element_code_id: Integer,
    year: Integer,
    value: Float,
    unit: String,
    flag_id: Integer
}]->(:Item)
```

HAS_PRICE (from prices)

sql

```
(:Country)-[pr:HAS_PRICE {
    item_code_id: Integer,
    element_code_id: Integer,
    year: Integer,
    months_code: String,
    value: Float,
    unit: String,
    flag_id: Integer
}]->(:Item)
```

Migration Implementation

Reference Data Migration

sql

```sql
-- Migrate Countries
SELECT * FROM cypher('fao', $$
    CREATE (c:Country {
        sql_id: a.id,
        area_code: a.area_code,
        name: a.area,
        m49_code: a.area_code_m49,
        source_dataset: a.source_dataset
    })
$$) AS (result agtype)
FROM area_codes a
WHERE a.area_code NOT LIKE 'aquastat-%';

-- Migrate Items
SELECT * FROM cypher('fao', $$
    CREATE (i:Item {
        sql_id: i.id,
        item_code: i.item_code,
        name: i.item,
        cpc_code: i.item_code_cpc,
        fbs_code: i.item_code_fbs,
        sdg_code: i.item_code_sdg,
        source_dataset: i.source_dataset
    })
$$) AS (result agtype)
FROM item_codes i;

-- Migrate Elements
SELECT * FROM cypher('fao', $$
    CREATE (e:Element {
        sql_id: e.id,
        element_code: e.element_code,
        name: e.element,
        source_dataset: e.source_dataset
    })
$$) AS (result agtype)
FROM elements e;
```

Relationship Migration

sql

```
-- Create TRADES_WITH relationships from trade_detailed_trade_matrix
WITH trade_batch AS (
    SELECT
        t.*,
        rc.area_code as reporter_code,
        pc.area_code as partner_code
    FROM trade_detailed_trade_matrix t
    JOIN area_codes rc ON t.reporter_country_code_id = rc.id
    JOIN area_codes pc ON t.partner_country_code_id = pc.id
    WHERE t.year >= 2020  -- Batching by year for performance
    LIMIT 10000
)
SELECT * FROM cypher('fao', $$
    MATCH (reporter:Country {area_code: $reporter_code})
    MATCH (partner:Country {area_code: $partner_code})
    CREATE (reporter)-[t:TRADES_WITH {
        item_code_id: $item_code_id,
        element_code_id: $element_code_id,
        year: $year,
        value: $value,
        unit: $unit,
        flag_id: $flag_id
    }]->(partner)
$$, row_to_json(trade_batch)) AS (result agtype)
FROM trade_batch;

-- Create PRODUCES relationships from production_crops_livestock
WITH production_batch AS (
    SELECT
        p.*,
        ac.area_code,
        ic.item_code
    FROM production_crops_livestock p
    JOIN area_codes ac ON p.area_code_id = ac.id
    JOIN item_codes ic ON p.item_code_id = ic.id
    WHERE p.year >= 2020
    LIMIT 10000
)
SELECT * FROM cypher('fao', $$
    MATCH (country:Country {area_code: $area_code})
    MATCH (item:Item {item_code: $item_code})
    CREATE (country)-[p:PRODUCES {
        element_code_id: $element_code_id,
```

```
        year: $year,
        value: $value,
        unit: $unit,
        flag_id: $flag_id
    }]->(item)
$$, row_to_json(production_batch)) AS (result agtype)
FROM production_batch;
```

Query Patterns

Basic Graph Queries

sql

```
-- Find all countries that trade wheat
SELECT * FROM cypher('fao', $$
    MATCH (c1:Country)-[t:TRADES_WITH]->(c2:Country)
    MATCH (i:Item {item_code: '0111'})
    WHERE t.item_code_id = i.sql_id AND t.year = 2023
    RETURN DISTINCT c1.name as exporter, c2.name as importer
$$) as (exporter agtype, importer agtype);

-- Find production by country
SELECT * FROM cypher('fao', $$
    MATCH (c:Country {area_code: '840'})-[p:PRODUCES]->(i:Item)
    WHERE p.year = 2023
    RETURN i.name, p.value, p.unit
    ORDER BY p.value DESC
$$) as (item agtype, value agtype, unit agtype);
```

Mixed SQL and Graph Queries

```sql
-- Combine SQL aggregation with graph traversal
WITH production_stats AS (
    SELECT
        area_code_id,
        item_code_id,
        AVG(value) as avg_production
    FROM production_crops_livestock
    WHERE year BETWEEN 2020 AND 2023
    GROUP BY area_code_id, item_code_id
)
SELECT
    ps.avg_production,
    graph_result.trade_partners
FROM production_stats ps
CROSS JOIN LATERAL (
    SELECT * FROM cypher('fao', $$
        MATCH (c:Country {sql_id: $country_id})-[:TRADES_WITH]->(partner:Country)
        RETURN count(DISTINCT partner) as trade_partners
    $$, json_build_object('country_id', ps.area_code_id))
    AS (trade_partners agtype)
) graph_result;
```

Path Finding

```sql
-- Find trade paths between countries
SELECT * FROM cypher('fao', $$
    MATCH path = (origin:Country {area_code: '076'})-[:TRADES_WITH*1..3]->(dest:Country {area_c
    WHERE ALL(r IN relationships(path) WHERE r.year = 2023)
    RETURN path, length(path) as hops
    ORDER BY hops
    LIMIT 10
$$) as (path agtype, hops agtype);
```

Performance Optimization

Indexing Strategy

sql

```sql
-- Create indexes for graph properties
CREATE INDEX ON fao._ag_label_vertex USING GIN (properties);
CREATE INDEX ON fao._ag_label_edge USING GIN (properties);

-- Create specific property indexes
CREATE INDEX idx_country_area_code ON fao.country USING btree ((properties->>'area_code'));
CREATE INDEX idx_item_item_code ON fao.item USING btree ((properties->>'item_code'));
CREATE INDEX idx_trades_year ON fao.trades_with USING btree ((properties->>'year'));
```

Batch Processing

python

```python
def migrate_relationships_batch(table_name, batch_size=10000):
    """Migrate relationships in batches to manage memory"""
    offset = 0
    while True:
        query = f"""
        SELECT * FROM {table_name}
        WHERE year >= 2020
        LIMIT {batch_size} OFFSET {offset}
        """
        # Process batch
        offset += batch_size
```

Maintenance and Operations

Backup Strategy

bash

```bash
# Backup both schemas
pg_dump -h localhost -p 5433 -U postgres \
  -n public -n fao -n ag_catalog \
  fao_graph > fao_graph_backup.sql
```

Statistics Update

```sql
-- Update graph statistics
ANALYZE fao._ag_label_vertex;
ANALYZE fao._ag_label_edge;

-- Vacuum for performance
VACUUM ANALYZE;
```

Monitoring Queries

```sql
-- Count nodes and relationships
SELECT * FROM cypher('fao', $$
    MATCH (n)
    RETURN labels(n)[0] as label, count(n) as count
$$) as (label agtype, count agtype);

-- Check relationship distribution
SELECT * FROM cypher('fao', $$
    MATCH ()-[r]->()
    RETURN type(r) as relationship, count(r) as count
$$) as (relationship agtype, count agtype);
```

API Integration

Query Function for API Endpoints

python

```python
from typing import List, Dict, Any
import asyncpg

async def execute_graph_query(
    cypher_query: str,
    parameters: Dict[str, Any] = None
) -> List[Dict[str, Any]]:
    """Execute Cypher query and return results"""

    sql_query = f"""
    SELECT * FROM cypher('fao', $${cypher_query}$$, $1::jsonb)
    AS result(data agtype)
    """

    async with get_db_connection() as conn:
        rows = await conn.fetch(sql_query, parameters or {})
        return [parse_agtype(row['data']) for row in rows]
```

View Creation for Common Patterns

sql

```sql
CREATE OR REPLACE VIEW trade_network AS
SELECT
    (properties->>'area_code')::text as reporter_code,
    (properties->>'name')::text as reporter_name,
    (end_properties->>'area_code')::text as partner_code,
    (end_properties->>'name')::text as partner_name,
    (properties->>'year')::integer as year,
    (properties->>'value')::float as value
FROM fao._ag_label_edge
WHERE label = 'TRADES_WITH';
```

Best Practices

1. **Use Foreign Tables for Live Data**: Keep source data in PostgreSQL, reference via foreign tables
2. **Batch Large Migrations**: Process relationships in year/country batches
3. **Index Strategically**: Create indexes on frequently queried properties
4. **Monitor Memory Usage**: AGE shares PostgreSQL memory allocation
5. **Regular Maintenance**: Run VACUUM and ANALYZE periodically

6. **Leverage Hybrid Queries**: Combine SQL aggregations with graph traversals

Conclusion

Apache AGE provides a powerful graph layer for the FAO database while maintaining the reliability and features of PostgreSQL. This implementation enables complex relationship queries, path analysis, and network algorithms without duplicating data or managing separate systems.