Storage Architecture - Tri-Store Hybrid System

Overview

Project Aura uses a **tri-store architecture** that leverages the strengths of three different storage systems:

1. **PostgreSQL** - Structured financial data
2. **MongoDB** - Semi-structured metadata and audit logs
3. **File System** - Unstructured data and binary files

This hybrid approach optimizes for performance, flexibility, and developer experience.

Storage Decision Matrix

Data Type	PostgreSQL	MongoDB	Files	Rationale
GL Account Balances	☑ Primary		Parquet cache	Structured, requires joins, SQL analytics
User Information	☑ Primary			Relational, foreign keys
Responsibility Matrix	☑ Primary			Many-to-many relationships
Review Status	☑ Primary			Transactional updates
Supporting Doc Metadata		☑ Primary		Flexible schema, varies per GL
Actual PDFs/Excel			☑ Primary	Binary data, large files
Comments & Annotations		☑ Primary		Nested structure, replies
Audit Trail Events	☑ Summary	☑ Detailed		Both: Summary in PG, full log in Mongo
Validation Results	☑ Pass/Fail	☑ Full Report	JSON backup	PG for status, Mongo for details
Vector Embeddings			☑ ChromaDB	Specialized vector storage
ML Model Artifacts			☑ MLflow	Binary model files
Raw CSV Uploads			☑ Landing zone	Temporary storage before parsing
Processed DataFrames			☑ Parquet	Fast columnar access

Data Flow Diagrams

1. CSV Upload Flow

```
User uploads CSV (Streamlit)
    ↓
Save to data/raw/ (File System)
    ↓
Parse & Validate Schema
    ↓
Load to PostgreSQL (gl_accounts table)
    ↓
Log metadata to MongoDB (upload_tracking)
    ↓
Cache as Parquet in data/processed/ (File System)
```

2. Supporting Document Flow

```
User uploads PDF/Excel
    ↓
Save binary to data/supporting_docs/{gl_code}/ (File System)
    ↓
Save metadata to MongoDB (supporting_docs collection)
    {
        gl_code: "1001",
        files: [{name, path, uploaded_by, timestamp}],
        comments: []
    }
    ↓
Log audit event to MongoDB (audit_trail)
    ↓
Update review_log in PostgreSQL
```

3. Validation Flow

```
Trigger validation (Button click)
    ↓
Load data from PostgreSQL
    ↓
Run Great Expectations suite
    ↓
Save pass/fail to PostgreSQL (review_log)
    ↓
Save full results to MongoDB (validation_results)
    ↓
Save checkpoint to data/processed/ (File System)
```

4. Analytics Query Flow

```
User requests dashboard
    ↓
Check Parquet cache in data/processed/
    ├─ HIT: Load from Parquet
    └─ MISS: Query PostgreSQL
        ↓
        Compute aggregations (SQL)
        ↓
        Cache results as Parquet
    ↓
Generate Plotly charts
    ↓
Display in Streamlit
```

5. RAG Chatbot Flow

```
User asks question
    ↓
Embed query using OpenAI
    ↓
Vector search in ChromaDB (data/vectors/)
    ↓
Retrieve relevant context
    ↓
Agent decides: Call tool OR use context
    ├─ Tool call: Query PostgreSQL for live data
    └─ Context: Answer from FAQ docs
    ↓
Generate response with LangChain
    ↓
Log interaction to MongoDB (audit_trail)
```

PostgreSQL Schema

Tables

users

```
id SERIAL PRIMARY KEY
name VARCHAR(255) NOT NULL
email VARCHAR(255) UNIQUE NOT NULL
department VARCHAR(100)
```

```
role VARCHAR(50)
created_at TIMESTAMP DEFAULT NOW()
updated_at TIMESTAMP DEFAULT NOW()
```

gl_accounts

```
id SERIAL PRIMARY KEY
account_code VARCHAR(50) NOT NULL
account_name VARCHAR(255) NOT NULL
entity VARCHAR(255) NOT NULL
balance DECIMAL(18, 2) NOT NULL
period VARCHAR(20) NOT NULL
assigned_user_id INTEGER REFERENCES users(id)
review_status VARCHAR(50) DEFAULT 'pending'
created_at TIMESTAMP DEFAULT NOW()
updated_at TIMESTAMP DEFAULT NOW()
UNIQUE(account_code, entity, period)
```

responsibility_matrix

```
id SERIAL PRIMARY KEY
gl_code VARCHAR(50) NOT NULL
user_id INTEGER REFERENCES users(id)
role VARCHAR(50) NOT NULL
created_at TIMESTAMP DEFAULT NOW()
UNIQUE(gl_code, user_id, role)
```

review_log

```
id SERIAL PRIMARY KEY
gl_account_id INTEGER REFERENCES gl_accounts(id)
reviewer_id INTEGER REFERENCES users(id)
status VARCHAR(50) NOT NULL
comments TEXT
reviewed_at TIMESTAMP DEFAULT NOW()
```

Indexes

- `idx_gl_accounts_period` on `gl_accounts(period)`
- `idx_gl_accounts_entity` on `gl_accounts(entity)`
- `idx_gl_accounts_status` on `gl_accounts(review_status)`
- `idx_review_log_gl` on `review_log(gl_account_id)`

MongoDB Collections

`supporting_docs`

```
{
  _id: ObjectId,
  gl_code: "1001",
  period: "2025-01",
  entity: "Adani Ports",
  files: [
    {
      name: "invoice_jan.pdf",
      path: "data/supporting_docs/1001/invoice_jan.pdf",
      uploaded_by: "john@adani.com",
      uploaded_at: ISODate("2025-01-15")
    }
  ],
  comments: [
    {
      user: "Manager",
      text: "Please clarify this variance",
      timestamp: ISODate("2025-01-16"),
      replies: []
    }
  ],
  created_at: ISODate,
  updated_at: ISODate
}
```

Indexes:

- `gl_code`
- `(gl_code, period)` compound

`audit_trail`

```
{
  _id: ObjectId,
  gl_code: "1001",
  action: "reviewed",
  actor: {
    user_id: "U123",
    name: "Jane Smith",
    email: "jane@adani.com"
  },
  details: {
    previous_status: "pending",
    new_status: "approved"
  },
```

```
  timestamp: ISODate
}
```

Indexes:

- `gl_code`
- `timestamp` (descending)
- `(gl_code, timestamp)` compound

`validation_results`

```
{
  _id: ObjectId,
  gl_code: "1001",
  period: "2025-01",
  validation_suite: "trial_balance_suite",
  results: {
    expectations: [...],
    success_count: 45,
    failure_count: 2,
    failures: [...]
  },
  passed: false,
  validated_at: ISODate
}
```

Indexes:

- `gl_code`
- `validation_suite`

File System Structure

```
data/
├── raw/                          # Landing zone for CSV uploads
│   ├── trial_balance_jan.csv
│   └── trial_balance_feb.csv
│
├── processed/                    # Cached Parquet files
│   ├── gl_accounts_2025-01.parquet
│   └── analytics_summary.parquet
│
├── supporting_docs/              # Organized by GL code
│   ├── 1001/
│   │   ├── invoice_jan.pdf
│   │   └── reconciliation.xlsx
│   └── 2005/
```

```
│       └── backup_working.pdf
│
├── vectors/                      # ChromaDB persistence
│   └── chroma.sqlite3
│
└── sample/                       # Sample data for testing
    └── trial_balance_sample.csv
```

Technology Stack

Database Drivers

- **psycopg2-binary** 2.9.9 - PostgreSQL adapter
- **SQLAlchemy** 2.0.30 - ORM and query builder
- **pymongo** 4.6.2 - MongoDB driver
- **pyarrow** 15.0.0 - Parquet file support

Storage Libraries

- **pandas** - DataFrame operations
- **chromadb** - Vector storage for RAG
- **mlflow** - ML model artifact storage

Connection Management

PostgreSQL

```
from src.db import get_postgres_session

session = get_postgres_session()
users = session.query(User).all()
session.close()
```

MongoDB

```
from src.db import get_mongo_database

db = get_mongo_database()
collection = db["supporting_docs"]
doc = collection.find_one({"gl_code": "1001"})
```

File Storage

```python
from src.db.storage import save_raw_csv, load_processed_parquet

# Save CSV
save_raw_csv(df, "trial_balance.csv")

# Load Parquet cache
cached_df = load_processed_parquet("analytics_summary")
```

Environment Variables

```
# PostgreSQL
POSTGRES_HOST=localhost
POSTGRES_PORT=5432
POSTGRES_DB=finnovate
POSTGRES_USER=admin
POSTGRES_PASSWORD=hackathon2025

# MongoDB
MONGO_HOST=localhost
MONGO_PORT=27017
MONGO_DB=finnovate
```

Docker Setup

Start databases:

```
docker-compose up -d
```

Check status:

```
docker-compose ps
```

View logs:

```
docker-compose logs -f postgres
docker-compose logs -f mongodb
```

Initialization

Run bootstrap script to:

1. Start Docker containers
2. Create PostgreSQL tables
3. Initialize MongoDB collections with indexes
4. Create file system directories

```
.\scripts\bootstrap.ps1
```

Performance Optimizations

1. **PostgreSQL Indexes** - Fast queries on period, entity, status
2. **Parquet Caching** - Columnar format for analytics
3. **MongoDB Compound Indexes** - Fast lookups on (gl_code, period)
4. **Connection Pooling** - Reuse database connections
5. **ChromaDB Persistence** - Avoid re-embedding on startup

Backup & Recovery

PostgreSQL

```
docker exec finnovate-postgres pg_dump -U admin finnovate > backup.sql
```

MongoDB

```
docker exec finnovate-mongodb mongodump --out /backup
```

Files

Regular filesystem backups of `data/` directory.

Phase Alignment

- **Phase 1 (Day 1):** File storage + basic PostgreSQL (gl_accounts, users)
- **Phase 2 (Day 2):** Full PostgreSQL schema + MongoDB (validation_results)
- **Phase 3 (Day 3):** MongoDB audit_trail + supporting_docs
- **Phase 4 (Day 4):** ChromaDB vector store for RAG
- **Phase 5 (Day 5-6):** Performance tuning, caching, observability