Database Initialization Guide

🎯 How PostgreSQL Initialization Works

PostgreSQL's official Docker image has a special feature:

Any `.sql` or `.sh` files in `/docker-entrypoint-initdb.d/` are automatically executed when the database is first created.

Our setup uses this feature:

```yaml
# In docker-compose.yml
postgres:
  volumes:
    - ./backend/init-db:/docker-entrypoint-initdb.d
```

This mounts our `init.sql` file into the container's initialization directory.

⚠️ Important: First-Time Only!

The initialization scripts ONLY run on the first startup when the database is empty.

If you've already run `docker-compose up` before, the database volume exists and init scripts won't run again.

🔍 Checking If Database Is Initialized

After starting containers, verify the database:

```bash
# Make script executable
chmod +x check-db.sh

# Run verification
./check-db.sh
```

Expected output if initialized correctly:

```
✓ PostgreSQL container is running

Checking database tables...

✓ Table 'users' exists
✓ Table 'ai_providers' exists
✓ Table 'api_keys' exists
✓ Table 'projects' exists
✓ Table 'chat_sessions' exists
✓ Table 'messages' exists
✓ Table 'permissions' exists

✅ Database is properly initialized!

📋 AI Providers in database: 4
```

🛠️ Troubleshooting Database Initialization

Problem 1: Database Already Existed

Symptom: Tables don't exist, but container is running.

Cause: You ran `docker-compose up` before creating `init.sql`, so the database volume was created empty.

Solution A - Full Reset (destroys all data):

```bash
# Stop and remove volumes
docker-compose down -v

# Start fresh
docker-compose up -d

# Wait 10 seconds for init
sleep 10

# Verify
./check-db.sh
```

Solution B - Manual initialization (keeps data if any):

```bash
chmod +x init-db-manual.sh
./init-db-manual.sh
```

Problem 2: Permission Errors

Symptom: Container logs show "permission denied" for init.sql

Solution:

```bash
# Fix file permissions
chmod 644 backend/init-db/init.sql

# Restart
docker-compose restart postgres
```

Problem 3: SQL Syntax Errors

Symptom: Some tables exist, others don't

Solution:

```bash
# Check PostgreSQL logs
docker-compose logs postgres

# Look for SQL errors
# Fix init.sql if needed
# Then run manual init
./init-db-manual.sh
```

📋 Manual Database Initialization

If automatic initialization didn't work, use the manual script:

```bash
# Make script executable
chmod +x init-db-manual.sh

# Run manual initialization
./init-db-manual.sh
```

This script:

1. Checks if PostgreSQL is running
2. Checks if init.sql exists
3. Asks for confirmation
4. Executes init.sql directly
5. Verifies the results

🔄 Database Reset Workflow

When you need a fresh start:

```bash
# 1. Stop all containers
docker-compose down

# 2. Remove volumes (WARNING: deletes all data!)
docker-compose down -v

# 3. Verify init.sql exists and is correct
cat backend/init-db/init.sql

# 4. Start containers
docker-compose up -d

# 5. Wait for initialization
echo "Waiting for database initialization..."
sleep 10

# 6. Check if database initialized
./check-db.sh

# 7. If not initialized, run manual script
./init-db-manual.sh
```

🧪 Testing Database Connection

From Host Machine

```bash
# Connect to database
docker-compose exec postgres psql -U aiplatform -d aiplatform

# List tables
\dt

# Check providers
SELECT * FROM ai_providers;

# Exit
\q
```

From Backend Code

```bash
# Check backend logs
docker-compose logs backend

# Should see:
# "Database connected successfully"
# "Server running on port 3000"
```

Via API

```bash
# Health check (doesn't need DB)
curl http://localhost/api/health

# Register user (needs DB)
curl -X POST http://localhost/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{"username":"test","password":"test123"}'

# If you get user_id and token back, database is working!
```

📝 Database Files Explained

init.sql Structure

```sql
-- 1. Create tables (IF NOT EXISTS prevents errors on re-run)
CREATE TABLE IF NOT EXISTS users (...);
CREATE TABLE IF NOT EXISTS ai_providers (...);
...

-- 2. Create indexes for performance
CREATE INDEX idx_projects_owner ON projects(owner_id);
...

-- 3. Insert default data (ON CONFLICT DO NOTHING prevents duplicates)
INSERT INTO ai_providers (...) VALUES (...)
ON CONFLICT (name) DO NOTHING;
```

The `IF NOT EXISTS` and `ON CONFLICT DO NOTHING` clauses make the script **idempotent** - you can run it multiple times safely.

🎯 Common Questions

Q: Do I need to manually create the database? A: No. The `POSTGRES_DB` environment variable in docker-compose.yml creates it automatically.

Q: What if I want to change the database schema? A:

1. Modify `init.sql`
2. Run `docker-compose down -v` (destroys data!)
3. Run `docker-compose up -d`
4. Or use database migrations (more advanced)

Q: Can I add more default data? A: Yes! Add more INSERT statements at the end of init.sql with `ON CONFLICT DO NOTHING`.

Q: How do I backup the database?

```bash
# Backup
docker-compose exec -T postgres pg_dump -U aiplatform aiplatform > backup.sql

# Restore
docker-compose exec -T postgres psql -U aiplatform -d aiplatform < backup.sql
```

🚀 Quick Start Checklist

- [ ] Create `backend/init-db/init.sql` (from artifact)
- [ ] Create `check-db.sh` (from artifact)
- [ ] Create `init-db-manual.sh` (from artifact)
- [ ] Make scripts executable: `chmod +x *.sh`
- [ ] Start containers: `docker-compose up -d`
- [ ] Wait 10 seconds
- [ ] Verify database: `./check-db.sh`
- [ ] If not initialized: `./init-db-manual.sh`
- [ ] Test API: `curl http://localhost/api/health`

🔐 Security Notes

For Production:

1. Change default passwords in docker-compose.yml
2. Don't expose PostgreSQL port 5432 to host
3. Use strong passwords in `.env`
4. Consider using Docker secrets instead of environment variables
5. Enable SSL for PostgreSQL connections
6. Restrict database user permissions