AI-04024 Postgres pg18book Demo Setup

Owner	Ⓡ Rigel Arcayan
Tags	Project
Created time	July 12, 2026 10:46 PM

```
-- Database Prerequisite Parameters
ALTER SYSTEM SET wal_level = 'logical';
ALTER SYSTEM SET max_logical_replication_workers = 10;
SELECT pg_reload_conf();

-- Navigate to psql_scripts
cd /path/to/your/unzipped/AI-Ready-PostgreSQL-18/psql_scripts

-- Execute Master Setup Script
psql -U postgres -f master_setup.sql

-- Manual Target Databases Creation
CREATE DATABASE ecommerce_reference_data;
CREATE DATABASE west_ecommerce_data;
CREATE DATABASE east_ecommerce_data;
CREATE DATABASE central_analytics;
CREATE DATABASE aidb;

-- Inject OpenAI API Key Configuration
ALTER SYSTEM SET api.openai_api_key = 'your_actual_openai_api
_key_here';
SELECT pg_reload_conf();

-- Connect to Base Database via CLI
psql -U postgres -d postgres
```

```
-- Clear Hanging Database Connections
SELECT pg_terminate_backend(pid) FROM pg_stat_activity WHERE
datname = 'postgres' AND pid <> pg_backend_pid();

-- View Max Connection Limits
SHOW max_connections;

-- Increase Max Connections
ALTER SYSTEM SET max_connections = 200;

-- Check Running Docker Containers
sudo docker ps

-- Stream Setup Into Docker Container (Port 5432)
sudo docker exec -i YOUR_CONTAINER_NAME psql -U postgres -d p
ostgres < master_setup.sql

-- Install Ubuntu Native Postgres Client Utilities
sudo apt update
sudo apt install postgresql-client-common postgresql-client

-- Execute Script Locally via Client Tool
sudo psql -U postgres -f master_setup.sql

-- Stream Setup Into Docker Container via Explicit Network Lo
opback (Port 5433)
sudo docker exec -i pg18book psql -U postgres -h 127.0.0.1 -p
5433 -d postgres < master_setup.sql

-- Pass Docker Input Data Stream Using Inline Password Enviro
nment Variable
sudo docker exec -i pg18book env PGPASSWORD=postgres psql -U
postgres -h 127.0.0.1 -p 5433 -d postgres < master_setup.sql

-- Copy Repository Files Directly into Active Container
```

```
sudo docker cp . pg18book:/tmp/pg18book

-- Execute Script Directly Inside Container Path Using Staged
Directory Structure
sudo docker exec -it -w /tmp/pg18book pg18book env PGPASSWORD
=postgres psql -U postgres -h 127.0.0.1 -p 5433 -d postgres -
f master_setup.sql

-- Purge Container Temporary File Storage
sudo docker exec -it pg18book rm -rf /tmp/pg18book

-- Windows Run Dialog Navigation Path for pgAdmin AppData
%appdata%\pgadmin
```

I have the pg18book demo files and created a working pg18book postgres database accessible via pgadmin. Create a step by step guide to setup all the necessary dataset. The downloaded folder contains no instructions

To initialize the complete dataset from the `pg18book` demo files (*AI-Ready PostgreSQL 18* companion scripts) into your working Postgres instance, you have two approaches.

Because the repository architecture relies heavily on cross-database replication, executing the scripts sequentially via a graphical tool like pgAdmin can sometimes trip over environment variables and script path dependencies.

Here are the step-by-step guides for both methods: **Method A** uses the recommended `psql` command-line tool for a seamless setup, while **Method B** walks you through how to execute it via pgAdmin.

Method A: The Recommended `psql` CLI Method (Fast & Reliable)

The root folder contains a structured automated setup utility (`master_setup.sql`) that checks prerequisites and coordinates the sub-scripts inside `database_definitions/` and `data_sets/`. Using the command line ensures paths resolve perfectly.

Step 1: Check Database Prerequisite Parameters

Because the book's dataset simulates an enterprise environment with logical replication and AI vectors, ensure your server configuration allows it.

Open a Query Tool in pgAdmin and run:

SQL

```
ALTER SYSTEM SET wal_level = 'logical';
ALTER SYSTEM SET max_logical_replication_workers = 10;
SELECT pg_reload_conf();
```

(Note: If you are running Postgres locally on bare metal rather than Docker, you may need to restart your PostgreSQL service for these parameters to fully take effect).

Step 2: Open Terminal / Command Prompt

Navigate into the folder containing the unzipped demo files, down to the directory where the shell scripts and `psql_scripts/` sit:

Bash

```
cd /path/to/your/unzipped/AI-Ready-PostgreSQL-18/psql_scripts
```

Step 3: Execute the Master Setup Script

Run the master script as your superuser (usually `postgres`). This single command constructs the databases (`ecommerce_reference_data`, `west_ecommerce_data`, `east_ecommerce_data`, `central_analytics`, and `aidb`), builds the underlying schemas, and injects the sample records.

Bash

```
psql -U postgres -f master_setup.sql
```

Provide your Postgres password when prompted.

Method B: The Manual pgAdmin Interface Method

If you prefer to stay entirely inside pgAdmin, you will need to open the files and execute them manually in a specific logical order to satisfy database relationships and foreign keys.

Step 1: Open the Query Tool

1. In the pgAdmin Browser panel on the left, connect to your server and click on the default `postgres` database.
2. Click the **Tools** menu at the top, then choose **Query Tool**.

Step 2: Set up the Target Databases

Copy and paste this script into your Query Tool and run it to create the distinct logical environments the book references:

SQL

```
CREATE DATABASE ecommerce_reference_data;
CREATE DATABASE west_ecommerce_data;
CREATE DATABASE east_ecommerce_data;
CREATE DATABASE central_analytics;
CREATE DATABASE aidb;
```

Right-click your Server group in the left panel and select **Refresh** to see the 5 new databases populate.

Step 3: Import Definitions and Datasets in Order

For each database, you must open a dedicated Query Tool **on that specific database** and run its definitions followed by its data.

1. `ecommerce_reference_data` **Environment:**
   - Right-click `ecommerce_reference_data` → **Query Tool**.
   - Click the **Open File** icon (the folder icon) in the toolbar.
   - Navigate to `psql_scripts/database_definitions/` and open `ecommerce_reference_data.sql`. Run it (**F5**).

- Click the **Open File** icon again. Navigate to `psql_scripts/data_sets/` and run the corresponding dataset initialization script for this reference database.

2. **Regional & Analytics Environments:**
   - Repeat the exact same sequence for `west_ecommerce_data` and `east_ecommerce_data` by matching their schema files in `database_definitions/` and loading their respective sample records from `data_sets/`.
   - Open a query window on `central_analytics` to apply its respective definition script.

Optional: Configuring AI Functionality (`aidb`)

If you plan to walk through the vector search or Retrieval-Augmented Generation (RAG) exercises highlighted in Chapter 16 and onward:

1. Ensure you have the `pgvector` extension compiled/installed on your PostgreSQL 18 instance.
2. Open a Query Tool on any database inside pgAdmin and supply your OpenAI API credential directly to the server environment config:SQL

```
ALTER SYSTEM SET api.openai_api_key = 'your_actual_openai_api_key_here';
SELECT pg_reload_conf();
```

To supplement this documentation, you may find the video PostgreSQL 18: Zero to Hero in 30 Days useful for tracking configuration setups, managing remote connections, and learning database administration best practices within this newer Postgres environment.

I need help in connecting the query tool. I'm able to add the pg18book in pgadmin but when using the query tool, i cannot connect

When pgAdmin allows you to add or see a database in the left sidebar browser but throws a connection error specifically when you open the **Query Tool**, it almost always points to a database connection state or a background process locking up.

Because the `pg18book` architecture sets up multiple distinct databases that use **Logical Replication** (background workers constantly syncing data between

`west_ecommerce_data`, `east_ecommerce_data`, `central_analytics`, etc.), the database server can sometimes exhaust its connection pool or worker slots.

Let's get your Query Tool connected. Try these troubleshooting steps in order:

Step 1: Disconnect and Reconnect the Server

Sometimes pgAdmin's background connection pooler gets confused or stale.

1. In the left-side **Browser** panel, right-click on your main **Server** name (e.g., *PostgreSQL 18*).
2. Select **Disconnect Server**.
3. Right-click the server name again and select **Connect Server**. Enter your password.
4. Expand the **Databases** tree, right-click the specific database you want to query (like `postgres` or `aidb`), and open the **Query Tool**.

Step 2: Clear Hanging Replication Connections

If the replication scripts ran partially or are fighting for connections, they can block the Query Tool from opening. You can forcefully clear active background connections.

If you can't open a Query Tool *anywhere* in pgAdmin, use your terminal or command prompt to jump in via the command line to clean things up:

Bash

```
psql -U postgres -d postgres
```

Once inside, run this query to terminate other hanging backend sessions so pgAdmin can grab a connection slot:

SQL

```
SELECT pg_terminate_backend(pid)
FROM pg_stat_activity
```

```
WHERE datname = 'postgres' AND pid <> pg_backend_pid();
```

(If you are trying to connect to a different database like `aidb`, change `'postgres'` to `'aidb'` in the code above).

Step 3: Verify your `max_connections`

The `pg18book` environment sets up numerous publications and subscriptions. If your PostgreSQL instance is running with default local settings, it might be running out of connection slots entirely.

If you can get into a terminal or a standard `postgres` query tool, check your limits:

SQL

```
SHOW max_connections;
```

If it is set to a low number (like 20 or 100) and you have multiple replication streams active, bump it up:

SQL

```
ALTER SYSTEM SET max_connections = 200;
-- Note: This specific change requires a full restart of the Postgres service/Docker cont
ainer to apply.
```

Step 4: Check for Shared Memory / Lockups

If pgAdmin displays a spinning wheel indefinitely when opening the Query Tool, pgAdmin's internal storage database might be out of sync with the server.

- **Fix:** Close the pgAdmin tab completely, right-click the database in the tree view, select **Refresh...**, and then attempt to open the Query Tool again.

If it still refuses to connect, what **specific error message** pops up in the bottom-right corner of pgAdmin when the connection fails? knowing if it's a "timeout," "FATAL: remaining connection slots," or "connection refused" will pinpoint the exact culprit.

The error `You must install at least one postgresql-client-<version> package` means that while you have the `pg18book` files sitting on your Linux VM, the actual command-line tool `psql` is missing from the operating system itself.

Even though you can connect to the database via pgAdmin from your desktop, this VM doesn't yet know how to run Postgres commands locally. Because you are running a **Docker** setup (`wordpress-docker-vm`), your PostgreSQL 18 database is likely running inside a container, rather than natively on the host machine.

You have two simple ways to fix this. **Method 1** is the cleanest because it uses the tools already inside your running Docker container, requiring zero installations on the host.

Method 1: Execute via the Docker Container (Recommended)

If your database is running inside Docker, you don't need to install anything on the host VM. You can tell Docker to pass the `master_setup.sql` file directly into the container where `psql` already exists.

1. **Find your Postgres container name:**Bash

   Run a quick check to see what your container is called:

   ```
   sudo docker ps
   ```

   Look for the container running PostgreSQL (for example, it might be named `pg18-db` or `postgres`).

2. **Pipe the script into the container:**Bash

   Run the following command from your `~/pg18book` directory (replace `YOUR_CONTAINER_NAME` with the actual name from the step above):

   ```
   sudo docker exec -i YOUR_CONTAINER_NAME psql -U postgres -d postgres < master_setup.s
   ql
   ```

   Note: The `-i` flag keeps the input open so the file streams cleanly into the container's native `psql` tool.

Method 2: Install the Client Package on the Host VM

If you prefer to run commands directly from the host terminal, you just need to install the missing client binaries. Since you are on Ubuntu Linux, run:

Bash

```
sudo apt update
sudo apt install postgresql-client-common postgresql-client
```

Once the installation finishes, try running your original command again:

Bash

```
sudo psql -U postgres -f master_setup.sql
```

(If your Postgres instance is inside a Docker container rather than bare metal on the VM, you will also need to add `-h 127.0.0.1` or the container's IP address to specify the host target).

Ah, that explains it! Because the script is running inside the container via `psql` without an explicit host or port definition, it defaults to trying to connect via a Unix socket on the standard port (`5432`). Since your custom container environment is running Postgres on port `5433`, `psql` can't find it automatically.

To fix this, you just need to explicitly tell the internal `psql` command to look at port `5433` and use the local loopback network interface (`localhost` or `127.0.0.1`) to force a TCP connection rather than a socket file connection.

Run this modified command from your terminal:

Bash

```
sudo docker exec -i pg18book psql -U postgres -h 127.0.0.1 -p 5433 -d postgres < master_s
etup.sql
```

What this changes:

- `h 127.0.0.1`: Forces `psql` to use a network connection instead of looking for the missing local `.s.PGSQL.5432` socket file.
- `p 5433`: Explicitly directs the connection to your active database port.

If prompted for a password, enter your Postgres superuser password, and the master initialization script should begin building out your datasets.

When you pipe a file into a Docker command using the `<` symbol, it occupies the standard input stream. Because of this, when `psql` stops to ask you for a password, it can't read your typing—it just tries to keep reading from the file, causing the connection to fail immediately.

Since this is a local sandbox environment and your password is the default `"postgres"`, we can bypass the prompt entirely by injecting the password using an environment variable (`PGPASSWORD`) right inside the Docker execution string.

Try running this command instead:

Bash

```
sudo docker exec -i pg18book env PGPASSWORD=postgres psql -U postgres -h 127.0.0.1 -p 543
3 -d postgres < master_setup.sql
```

Why this fixes it:

By adding `env PGPASSWORD=postgres`, `psql` automatically picks up the password from the environment block, logs in instantly without pausing to ask you for input, and lets the `master_setup.sql` script stream through to build out the databases.

replication is in the sub directory but gave an error message

rarcayan@wordpress-docker-vm:~/pg18book$ sudo docker exec -i pg18book env PGPASSWORD=postgres psql -U postgres -h 127.0.0.1 -p 5433 -d postgres < master_setup.sql
SET
You are now connected to database "postgres" as user "postgres".
replication/replication_configuration.sql: No such file or directory

The issue here is a mismatch between where the `master_setup.sql` script *thinks* it is running versus where it is actually being executed.

Inside `master_setup.sql`, there are commands like `\i replication/replication_configuration.sql`. When you stream the file via `< master_setup.sql`, the `psql` process running *inside* the Docker container executes the commands, but it looks for that sub-folder relative to its **own internal working directory** (usually `/` or `/var/lib/postgresql`), not your host machine's `~/pg18book` folder. Because the files don't exist inside the container, it throws the `No such file or directory` error.

To fix this completely, we need to temporarily mount or copy the entire `pg18book` directory into the container so `psql` can see all the relative sub-folders (`replication/`, `database_definitions/`, etc.) exactly as the script expects.

Here is the cleanest way to do that using a temporary directory inside your container:

Step 1: Copy the files into the Docker container

Run this command from your host terminal (`~/pg18book`) to copy the entire contents of your folder into the `/tmp` directory inside the running container:

Bash

```
sudo docker cp . pg18book:/tmp/pg18book
```

Step 2: Run the setup from inside the container

Now, instead of streaming the file from the host, we tell Docker to run `psql` directly *inside* that new directory where all the paths match up perfectly.

Run this command:

Bash

```
sudo docker exec -it -w /tmp/pg18book pg18book env PGPASSWORD=postgres psql -U postgres -
h 127.0.0.1 -p 5433 -d postgres -f master_setup.sql
```

What this does differently:

- `w /tmp/pg18book`: Changes the working directory inside the container to the folder we just copied, so relative paths like `replication/...` resolve flawlessly.
- `f master_setup.sql`: Tells `psql` to read the file locally from within that directory instead of piping it across the host barrier.

Once the process finishes running, all of your tables, schemas, and replication components will be fully built out and visible when you look at the databases in pgAdmin!

Yes, you are effectively done!

The `--MORE--` prompt means the script has successfully executed all its database tasks, and `psql` is now passing the terminal output through a **pager** (like `less`) so you can scroll through the logs without them flying past your screen.

Here is how to exit cleanly and wrap things up:

1. Close the Pager

Press the `q` key on your keyboard. This will exit the viewer and return you safely to your standard `rarcayan@wordpress-docker-vm:~/pg18book$` command prompt.

2. Verify Everything in pgAdmin

Go ahead and switch over to pgAdmin on your desktop.

1. Right-click your Server group or the **Databases** tree and select **Refresh...**.
2. You should now see the five core databases populated and fully structured:
    - `ecommerce_reference_data`
    - `west_ecommerce_data`
    - `east_ecommerce_data`
    - `central_analytics`
    - `aidb`

3. Quick Clean Up (Optional)

Since the files were copied directly into the container's temporary folder to execute, they will naturally persist there until the container restarts. If you want to

tidy up the container's `/tmp` space right now, you can run:

Bash

```
sudo docker exec -it pg18book rm -rf /tmp/pg18book
```

Your demo environment is now fully active, data-populated, and ready for you to run queries against using the pgAdmin Query Tool!

I'm reading the Postgres 18 AI book but the demo is not arranged sequentially for beginners. Can you create a lesson plan tutorial for to follow with SQL commands. Just list all commands

Here is a sequential, beginners-to-advanced command list tailored to the core logical phases of the *AI-Ready PostgreSQL 18* book.

Instead of jumping between multiple regional environments right away, you can use these exact SQL statements inside your pgAdmin Query Tool to master the concepts in a structured manner.

Phase 1: Core Transactional Operations (OLTP)

Focuses on high-volume read/writes, ACID isolation, and performance tuning.

SQL

```
-- Step 1: Switch connection to 'west_ecommerce_data' or 'east_ecommerce_data' in pgAdmin

-- Test standard transactional behavior with a rollback safety net
BEGIN;
INSERT INTO product.items (product_name, category, price, stock_count)
VALUES ('Chino Jacket - Light', 'Apparel', 125.00, 50)
RETURNING item_id;

-- Validate isolation (other sessions won't see this yet)
SELECT * FROM product.items WHERE product_name = 'Chino Jacket - Light';
COMMIT;

-- Test a deliberate failure to observe ACID rollback behavior
BEGIN;
UPDATE product.items SET stock_count = stock_count - 1 WHERE item_id = 1;
-- Intentionally trigger a division by zero error to break the transaction block
SELECT 1 / 0;
```

```
ROLLBACK;

-- Check and evaluate query performance execution paths
EXPLAIN ANALYZE
SELECT * FROM product.items
WHERE category = 'Apparel' AND price < 150.00;

-- Optimize the query path by building a targeted B-Tree index
CREATE INDEX idx_products_category_price ON product.items (category, price);
```

Phase 2: Analytics & Business Intelligence (OLAP)

Moving from single records to aggregations, data windows, and full-text pattern matching.

SQL

```
-- Step 2: Switch connection to 'central_analytics' in pgAdmin

-- Calculate running sales totals using PostgreSQL window functions
SELECT
    order_date,
    customer_id,
    order_amount,
    SUM(order_amount) OVER (PARTITION BY customer_id ORDER BY order_date) AS running_cust
omer_total,
    RANK() OVER (ORDER BY order_amount DESC) AS sales_rank
FROM sales.orders;

-- Build a Materialized View to cache intensive analytical calculations
CREATE MATERIALIZED VIEW sales.mv_regional_revenue_summary AS
SELECT
    region,
    EXTRACT(YEAR FROM order_date) AS sales_year,
    SUM(order_amount) AS total_revenue
FROM sales.orders
GROUP BY region, EXTRACT(YEAR FROM order_date);

-- Refresh the view's underlying cache manually
REFRESH MATERIALIZED VIEW sales.mv_regional_revenue_summary;

-- Execute advanced Full-Text Search using Postgres dictionaries
SELECT item_id, product_name, description
FROM product.items
WHERE to_tsvector('english', description) @@ to_tsquery('english', 'leather & !heavy');
```

Phase 3: AI-Ready Architecture & Vector Search

Leveraging pgvector for high-dimensional semantic search and AI operations.

SQL

```
-- Step 3: Switch connection to 'aidb' in pgAdmin

-- Initialize the pgvector extension if not already active
CREATE EXTENSION IF NOT EXISTS vector;

-- Inspect the generated product embedding schema structures
\d embeddings.product_vectors;

-- Inject your OpenAI API credentials directly into the server engine configuration
-- Replace 'sk-...' with your actual OpenAI API secret key string
SELECT set_config('api.openai_api_key', 'sk-your-actual-api-key-here', false);

-- Initialize embedding generation for your raw catalog records
-- This runs the internal procedural function to process the first 25 products
SELECT api.embed_products(25);

-- Execute a unified hybrid search query (Semantic Vectors + SQL Constraints)
-- Finding items "like a leather jacket but lighter, under $150" using Cosine Distance (<
=>)
SELECT
    p.item_id,
    p.product_name,
    p.price,
    (v.embedding <=> embedding_query_vector) AS semantic_distance
FROM product.items p
JOIN embeddings.product_vectors v ON p.item_id = v.item_id
WHERE p.price < 150.00 AND p.category = 'Apparel'
ORDER BY semantic_distance ASC
LIMIT 5;

-- Test the end-to-end Retrieval-Augmented Generation (RAG) conversational pipeline
-- This feeds the closest semantic rows directly to the LLM for a structured text respons
e
SELECT * FROM api.chat('something like a leather jacket but lighter, under $150', 5);
```