Connecting to PostgreSQL: psql and a Real Spring Boot DataSource

Run PostgreSQL locally with Docker, connect directly with psql, and see what this project's own Spring Boot DataSource configuration actually means at the PostgreSQL level. The 2nd lesson in the PostgreSQL Foundations category.

Beginner 15 min

Last updated: August 24, 2026
Read online: https://www.learnforgex.com/en/topics/connecting-to-postgresql

---

"PostgreSQL and the Relational Model" was entirely conceptual — no PostgreSQL was ever actually running. This lesson is the opposite: get a real PostgreSQL server running, connect to it directly with its own command-line client, and then look at exactly how this project's own Spring Boot configuration connects to that same kind of server underneath `TopicRepository` and every other repository call.

Running PostgreSQL Locally

The fastest way to get a real, disposable PostgreSQL server running, without installing anything system-wide, is Docker:

```
docker run --name postgres-learning -e POSTGRES_PASSWORD=learning -p 5433:5432 -d postgres:
16
```

`-e POSTGRES_PASSWORD=learning` sets the password for PostgreSQL's default `postgres` user. `-p 5433:5432` maps container port `5432` (PostgreSQL's real, standard port, inside the container) to port `5433` on your own machine — this exact mapping, `5433`, is not an arbitrary choice for this example: it's the real port this project's own `application-dev.yml` and `application-test.yml` already connect to, precisely so a locally installed PostgreSQL (which would normally claim `5432` for itself) never conflicts with this container. A native install (via your OS's package manager) works too and is a reasonable alternative — the rest of this lesson applies identically either way, since what matters is a PostgreSQL server listening on some port, not how it got started.

Connecting with psql

`psql` is PostgreSQL's own command-line client — the tool every PostgreSQL installation ships with, independent of any GUI tool or any Java code.

```
psql -h localhost -p 5433 -U postgres
```

`-h` is the host, `-p` the port, `-U` the username to connect as. After entering the password, you land at a `postgres=#` prompt — a direct, interactive connection to the server itself, with no Spring Boot, no Hibernate, and no JDBC driver anywhere in between.

A First Look Around: psql Meta-Commands

`psql` has its own commands, distinct from SQL itself, all starting with a backslash — worth knowing a handful of immediately from the very first session.

```
postgres=# \l
                                  List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    | ...
-----------+----------+----------+-------------+-------------+-----
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | ...
 learning  | learning | UTF8     | en_US.UTF-8 | en_US.UTF-8 | ...

postgres=# \c learning
You are now connected to database "learning" as user "postgres".

learning=# \dt
           List of relations
 Schema |   Name   | Type  |  Owner
--------+----------+-------+---------
 public | topic    | table | learning
 public | category | table | learning
 public | course   | table | learning

learning=# \d topic
                    Table "public.topic"
    Column      |  Type   | Collation | Nullable | Default
----------------+---------+-----------+----------+---------
 id             | bigint  |           | not null |
 category_id    | bigint  |           | not null |
 slug           | text    |           | not null |
 ...
```

`\l` lists every database on the server. `\c <database>` switches which database the current session is connected to — the `learning=#` prompt (instead of `postgres=#`) confirms the switch happened. `\dt` lists tables in the current database — this project's own real `topic` /

`category` / `course` tables, created by its Flyway migrations, are exactly what would show up here. `\d <table>` describes one table's columns — the exact same `topic` table walked through conceptually in "PostgreSQL and the Relational Model," now seen for real.

The JDBC URL: What jdbc:postgresql://host:port/database Actually Means

This project's own Spring Boot configuration never calls `psql` — but it connects using the exact same three pieces of information `psql`'s `-h` / `-p` flags and `\c` command just used.

A JDBC URL like `jdbc:postgresql://localhost:5433/learning` breaks down piece by piece: `jdbc:postgresql:` names the JDBC driver to use (PostgreSQL's own); `localhost:5433` is the host and port — identical in meaning to `psql -h localhost -p 5433`; `learning` is the database name — identical in meaning to `psql`'s `\c learning`. A username and password are supplied separately, alongside the URL, the same way `psql -U postgres` supplied one on the command line.

This Project's Own DataSource Configuration, and Connecting to the Same Database with psql

This project's real `application-dev.yml` configures exactly this:

```
spring:
  datasource:
    url: jdbc:postgresql://localhost:5433/learning
    username: learning
    password: learning
```

Connecting to that EXACT SAME database, by hand, with `psql`, needs nothing more than translating those three lines back into `psql`'s own flags:

```
psql -h localhost -p 5433 -U learning -d learning
```

`-d learning` connects directly to the `learning` database rather than the default `postgres` one, skipping the separate `\c learning` step. From this `psql` session, `\dt` shows the exact same tables Spring Boot's own `DataSource` reads from and writes to — there's no separate, hidden database Spring Boot uses; it's the identical PostgreSQL server and the identical `learning` database, just reached through a JDBC driver instead of `psql`.

> This project's `application-test.yml` points at a different database on the same server — `learning_test` instead of `learning`, same host, same port. Connecting to it directly is just `psql -h localhost -p 5433 -U learning -d learning_test` — the same three pieces of information, a different database name.

Where Spring Boot's DataSource Bean Actually Comes From

None of the three `application-dev.yml` lines above are read by application code directly — Spring Boot's auto-configuration is what turns them into a real, usable `DataSource` bean, exactly as "Spring Boot Auto-Configuration & Properties" already covered for auto-configuration in general, and as "JPA, Hibernate, and Spring Data JPA" already covered specifically for how that `DataSource` feeds into Hibernate and Spring Data JPA above it. Nothing new needs to be said about that mechanism here — this lesson's job was showing what those three configuration values actually mean, at the PostgreSQL level, not re-explaining how Spring Boot wires them up.

Common Misconceptions

"Spring Boot needs some special setup to talk to PostgreSQL." It doesn't — `spring.datasource.url` / `username` / `password` are the exact same three things `psql -h` / `-p` / `-U` / `-d` need; Spring Boot just carries them through a JDBC driver instead of a terminal session. **"`psql` and a GUI database tool are fundamentally different ways of talking to PostgreSQL."** They're not — both ultimately connect using the same host/port/database/credentials and speak the same PostgreSQL wire protocol; `psql` is simply the one that ships with PostgreSQL itself, with no separate installation. **"Port 5433 is PostgreSQL's real port."** It isn't — `5432` is PostgreSQL's standard port; `5433` is specifically this project's own choice (a Docker port mapping) to avoid colliding with a separately, locally installed PostgreSQL that would otherwise already be using `5432`.

Best Practices

- Learn `psql`'s basic meta-commands (`\l`, `\c`, `\dt`, `\d <table>`, `\q`) early — they're the fastest way to look at what's actually in a database, independent of any application code.
- When a Spring Boot application can't connect to PostgreSQL, try connecting to the exact same host/port/database/credentials with `psql` directly — it isolates whether the problem is the database itself or something in the Spring/JDBC layer above it.
- Keep a locally running PostgreSQL's port in mind when a project (like this one) intentionally uses a non-default port — it's almost always there to avoid colliding with another PostgreSQL instance, not an arbitrary choice.
- Treat a JDBC URL as three named pieces (driver, host:port, database name), not an opaque string — that's what makes translating one into `psql` flags (or vice versa) straightforward.

Common Mistakes

- Assuming a JDBC connection failure means something is wrong in Java/Spring code, without first checking whether `psql` itself can connect to the same host/port/database at all.

- Forgetting `-d <database>` when connecting with `psql` and being confused to land in the default `postgres` database instead of the one actually being worked with.
- Assuming PostgreSQL is always reachable on port `5432` without checking a project's own configuration first — plenty of real setups (this one included) intentionally use a different port.
- Treating `psql` as a lesser, "just for DBAs" tool instead of the fastest way to directly inspect exactly what a Spring Boot application's own `DataSource` is actually connected to.

Summary, Cheat Sheet, and Glossary

Summary

- Docker (`docker run ... postgres:16`) is the fastest way to get a real, disposable PostgreSQL server running locally; a native install works identically for the rest of this lesson.
- `psql` is PostgreSQL's own command-line client, connecting directly with `-h` / `-p` / `-U` / `-d`, independent of any Java code.
- `\l`, `\c`, `\dt`, and `\d <table>` are psql's own commands (not SQL) for listing databases, switching databases, listing tables, and describing a table's columns.
- A JDBC URL (`jdbc:postgresql://host:port/database`) encodes the exact same host, port, and database name `psql`'s flags use — plus a separately supplied username and password.
- This project's own `application-dev.yml` and a direct `psql` connection reach the identical PostgreSQL server and database — Spring Boot's auto-configuration (already covered elsewhere) is what turns those three YAML values into a real `DataSource` bean.

Cheat Sheet

```
# Run PostgreSQL locally with Docker
docker run --name postgres-learning -e POSTGRES_PASSWORD=learning -p 5433:5432 -d postgres:
16

# Connect with psql
psql -h localhost -p 5433 -U learning -d learning

# psql meta-commands
\l              -- list databases
\c <database>   -- switch database
\dt             -- list tables
\d <table>      -- describe a table's columns
\q              -- quit
```

```
# This project's application-dev.yml
spring:
  datasource:
    url: jdbc:postgresql://localhost:5433/learning
    username: learning
    password: learning
```

Glossary

- **psql**: PostgreSQL's own command-line client, connecting directly to a server independent of any application code.
- **Meta-command**: a `psql`-specific command (starting with `\`) like `\dt` or `\c`, distinct from SQL itself.
- **JDBC URL**: a connection string (`jdbc:postgresql://host:port/database`) encoding which driver, host, port, and database to connect to.
- **DataSource**: the Spring bean, auto-configured from `spring.datasource.*` properties, that application code (via Hibernate and Spring Data JPA) actually uses to reach PostgreSQL.