PostgreSQL'e Bağlanmak: psql ve Gerçek Bir Spring Boot DataSource

Docker ile yerel bir PostgreSQL çalıştır, psql ile doğrudan bağlan, ve bu projenin kendi Spring Boot DataSource yapılandırmasının PostgreSQL seviyesinde gerçekte ne anlama geldiğini gör. PostgreSQL Foundations kategorisinin 2.'si.

Başlangıç 15 dk

Son güncelleme: 24 Ağustos 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/connecting-to-postgresql

---

"PostgreSQL ve İlişkisel Model" tamamen kavramsaldı -- gerçekte hiçbir PostgreSQL hiç çalışmadı. Bu ders tam tersi: gerçek bir PostgreSQL sunucusunu çalıştırmak, ona kendi komut satırı istemcisiyle doğrudan bağlanmak, ve sonra bu projenin kendi Spring Boot yapılandırmasının, `TopicRepository`'nin ve diğer her repository çağrısının altında, tam olarak aynı türden bir sunucuya nasıl bağlandığına bakmak.

PostgreSQL'i Yerel Olarak Çalıştırmak

Sistem geneline hiçbir şey kurmadan gerçek, tek kullanımlık bir PostgreSQL sunucusunu çalıştırmanın en hızlı yolu Docker'dır:

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

`-e POSTGRES_PASSWORD=learning`, PostgreSQL'in varsayılan `postgres` kullanıcısının şifresini ayarlar. `-p 5433:5432`, container port `5432`'yi (PostgreSQL'in container içindeki gerçek, standart portu) kendi makinendeki `5433` portuna eşler -- tam olarak bu eşleme, `5433`, bu örnek için keyfi bir seçim değildir: bu, bu projenin kendi `application-dev.yml`'inin ve `application-test.yml`'inin zaten bağlandığı gerçek port, tam olarak yerel olarak kurulu bir PostgreSQL'in (normalde `5432`'yi kendisi için talep edecek olan) bu container'la asla çakışmaması için. Bir native kurulum (işletim sisteminin paket yöneticisi üzerinden) de işe yarar ve makul bir alternatiftir -- bu dersin geri kalanı ikisi için de birebir aynı şekilde geçerlidir, çünkü önemli olan bir portta dinleyen bir PostgreSQL sunucusudur, nasıl başlatıldığı değil.

psql ile Bağlanmak

`psql`, PostgreSQL'in kendi komut satırı istemcisidir -- her PostgreSQL kurulumunun, herhangi bir GUI aracından ya da herhangi bir Java kodundan bağımsız olarak birlikte geldiği araç.

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

`-h` host'tur, `-p` porttur, `-U` bağlanılacak kullanıcı adıdır. Şifreyi girdikten sonra, bir `postgres=#` prompt'una inersin -- sunucunun kendisine doğrudan, etkileşimli bir bağlantı, arada hiçbir Spring Boot, Hibernate, ya da JDBC driver'ı olmadan.

İlk Bir Bakış: psql Meta-Komutları

`psql`'in, SQL'in kendisinden ayrı, hepsi bir ters eğik çizgiyle başlayan kendi komutları var -- tam ilk oturumdan itibaren bir avucunu bilmeye değer.

```
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`, sunucudaki her veritabanını listeler. `\c <database>`, mevcut oturumun hangi veritabanına bağlı olduğunu değiştirir -- `learning=#` prompt'u (`postgres=#` yerine) değişimin gerçekleştiğini doğrular. `\dt`, mevcut veritabanındaki tabloları listeler -- bu projenin kendi gerçek `topic` / `category` / `course` tabloları, Flyway migration'ları tarafından oluşturulan, tam

olarak burada görünecek olan şeydir. `\d <table>`, bir tablonun sütunlarını tarif eder -- "PostgreSQL ve İlişkisel Model"de kavramsal olarak geçilen aynı `topic` tablosu, şimdi gerçekte görülüyor.

JDBC URL: jdbc:postgresql://host:port/database Gerçekte Ne Anlama Gelir

Bu projenin kendi Spring Boot yapılandırması `psql`'i hiç çağırmaz -- ama `psql`'in `-h`/`-p` bayraklarının ve `\c` komutunun az önce kullandığı tam olarak aynı üç bilgi parçasını kullanarak bağlanır.

`jdbc:postgresql://localhost:5433/learning` gibi bir JDBC URL'i parça parça ayrışır: `jdbc:postgresql:`, kullanılacak JDBC driver'ını adlandırır (PostgreSQL'in kendisi); `localhost:5433`, host ve porttur -- `psql -h localhost -p 5433` ile anlamca aynıdır; `learning`, veritabanı adıdır -- `psql`'in `\c learning`'i ile anlamca aynıdır. Bir kullanıcı adı ve şifre, URL'in yanında ayrı olarak sağlanır, tıpkı `psql -U postgres`'in komut satırında bir tane sağlaması gibi.

Bu Projenin Kendi DataSource Yapılandırması, ve Aynı Veritabanına psql ile Bağlanmak

Bu projenin gerçek `application-dev.yml`'i tam olarak bunu yapılandırıyor:

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

TAM OLARAK O AYNI veritabanına, elle, `psql` ile bağlanmak, bu üç satırı `psql`'in kendi bayraklarına geri çevirmekten fazlasına ihtiyaç duymaz:

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

`-d learning`, ayrı bir `\c learning` adımını atlayarak, varsayılan `postgres` veritabanı yerine doğrudan `learning` veritabanına bağlanır. Bu `psql` oturumundan, `\dt`, Spring Boot'un kendi `DataSource`'unun okuduğu ve yazdığı tam olarak aynı tabloları gösterir -- Spring Boot'un kullandığı ayrı, gizli bir veritabanı yoktur; birebir aynı PostgreSQL sunucusu ve birebir aynı `learning` veritabanıdır, yalnızca `psql` yerine bir JDBC driver'ı üzerinden ulaşılmış.

> Bu projenin `application-test.yml`'i aynı sunucudaki farklı bir veritabanını işaret eder -- `learning` yerine `learning_test`, aynı host, aynı port. Ona doğrudan bağlanmak yalnızca `psql -h localhost -p 5433 -U learning -d learning_test`'tir -- aynı üç bilgi parçası, farklı bir veritabanı adı.

Spring Boot'un DataSource Bean'i Gerçekte Nereden Gelir

Yukarıdaki üç `application-dev.yml` satırının hiçbiri uygulama kodu tarafından doğrudan okunmaz -- Spring Boot'un auto-configuration'ı, bunları gerçek, kullanılabilir bir `DataSource` bean'ine dönüştüren şeydir, tam olarak "Spring Boot Auto-Configuration ve Properties"in genel olarak auto-configuration için zaten işlediği, ve "JPA, Hibernate ve Spring Data JPA"nın özellikle o `DataSource`'un üzerindeki Hibernate'e ve Spring Data JPA'ya nasıl beslendiği için zaten işlediği gibi. Burada o mekanizma hakkında yeni bir şey söylemeye gerek yok -- bu dersin işi, Spring Boot'un onları nasıl bağladığını yeniden açıklamak değil, o üç yapılandırma değerinin, PostgreSQL seviyesinde, gerçekte ne anlama geldiğini göstermekti.

Yaygın Yanlış Anlamalar

"Spring Boot'un PostgreSQL'le konuşmak için özel bir kuruluma ihtiyacı vardır." Yoktur -- `spring.datasource.url` / `username` / `password`, `psql -h` / `-p` / `-U` / `-d`'nin ihtiyaç duyduğu tam olarak aynı üç şeydir; Spring Boot onları yalnızca bir terminal oturumu yerine bir JDBC driver'ı üzerinden taşır. **"`psql` ve bir GUI veritabanı aracı, PostgreSQL'le konuşmanın temelde farklı yollarıdır."** Değildir -- ikisi de nihayetinde aynı host/port/veritabanı/kimlik bilgileriyle bağlanır ve aynı PostgreSQL wire protokolünü konuşur; `psql`, yalnızca ayrı bir kurulum gerektirmeden PostgreSQL'in kendisiyle birlikte gelen olanıdır. **"5433 portu PostgreSQL'in gerçek portudur."** Değildir -- `5432`, PostgreSQL'in standart portudur; `5433`, özellikle bu projenin kendi seçimidir (bir Docker port eşlemesi), aksi hâlde zaten `5432`'yi kullanıyor olacak, ayrı, yerel olarak kurulu bir PostgreSQL'le çakışmamak için.

Best Practices

- `psql`'in temel meta-komutlarını (`\l`, `\c`, `\dt`, `\d <table>`, `\q`) erken öğren -- bunlar, herhangi bir uygulama kodundan bağımsız olarak, bir veritabanında gerçekte ne olduğuna bakmanın en hızlı yolu.
- Bir Spring Boot uygulaması PostgreSQL'e bağlanamadığında, tam olarak aynı host/port/veritabanı/kimlik bilgileriyle doğrudan `psql` ile bağlanmayı dene -- sorunun veritabanının kendisinde mi yoksa üzerindeki Spring/JDBC katmanında bir şeyde mi olduğunu izole eder.
- Bir proje (bunun gibi) bilinçli olarak varsayılan olmayan bir port kullandığında, yerel olarak çalışan bir PostgreSQL'in portunu aklında tut -- bu neredeyse her zaman başka bir PostgreSQL instance'ıyla çakışmamak içindir, keyfi bir seçim değil.
- Bir JDBC URL'ini opak bir string değil, üç adlandırılmış parça (driver, host:port, veritabanı adı) olarak ele al -- bir tanesini `psql` bayraklarına (ya da tam tersine) çevirmeyi basit kılan şey budur.

Yaygın Hatalar

- Önce `psql`'in kendisinin aynı host/port/veritabanına hiç bağlanıp bağlanamadığını kontrol etmeden, bir JDBC bağlantı hatasının Java/Spring kodunda bir şeyin yanlış olduğu anlamına geldiğini varsaymak.
- `psql` ile bağlanırken `-d <database>`'i unutup, gerçekte çalışılan veritabanı yerine varsayılan `postgres` veritabanına inince kafa karışıklığı yaşamak.
- Önce bir projenin kendi yapılandırmasını kontrol etmeden, PostgreSQL'in her zaman `5432` portunda erişilebilir olduğunu varsaymak -- birçok gerçek kurulum (bu dahil) bilinçli olarak farklı bir port kullanır.
- `psql`'i, bir Spring Boot uygulamasının kendi `DataSource`'unun gerçekte neye bağlı olduğunu doğrudan incelemenin en hızlı yolu yerine, "yalnızca DBA'lar için" daha aşağı bir araç olarak ele almak.

Özet, Cheat Sheet ve Terimler Sözlüğü

Özet

- Docker (`docker run ... postgres:16`), yerel olarak gerçek, tek kullanımlık bir PostgreSQL sunucusunu çalıştırmanın en hızlı yolu; bir native kurulum, bu dersin geri kalanı için birebir aynı şekilde işler.
- `psql`, PostgreSQL'in kendi komut satırı istemcisidir, herhangi bir Java kodundan bağımsız olarak doğrudan `-h` / `-p` / `-U` / `-d` ile bağlanır.
- `\l`, `\c`, `\dt`, ve `\d <table>`, veritabanlarını listelemek, veritabanı değiştirmek, tabloları listelemek, ve bir tablonun sütunlarını tarif etmek için psql'in kendi komutlarıdır (SQL değil).
- Bir JDBC URL'i (`jdbc:postgresql://host:port/database`), `psql`'in bayraklarının kullandığı tam olarak aynı host, port, ve veritabanı adını -- artı ayrı sağlanan bir kullanıcı adı ve şifreyi -- kodlar.
- Bu projenin kendi `application-dev.yml`'i ve doğrudan bir `psql` bağlantısı, birebir aynı PostgreSQL sunucusuna ve veritabanına ulaşır -- Spring Boot'un auto-configuration'ı (başka yerde zaten işlenen), o üç YAML değerini gerçek bir `DataSource` bean'ine dönüştüren şeydir.

Cheat Sheet

```
# PostgreSQL'i Docker ile yerel olarak çalıştır
docker run --name postgres-learning -e POSTGRES_PASSWORD=learning -p 5433:5432 -d postgres:
16

# psql ile bağlan
psql -h localhost -p 5433 -U learning -d learning

# psql meta-komutları
\l              -- veritabanlarını listele
\c <database>   -- veritabanı değiştir
\dt             -- tabloları listele
\d <table>      -- bir tablonun sütunlarını tarif et
\q              -- çık
```

```
# Bu projenin application-dev.yml'i
spring:
  datasource:
    url: jdbc:postgresql://localhost:5433/learning
    username: learning
    password: learning
```

Terimler Sözlüğü

- **psql**: herhangi bir uygulama kodundan bağımsız olarak bir sunucuya doğrudan bağlanan, PostgreSQL'in kendi komut satırı istemcisi.
- **Meta-komut**: `\dt` ya da `\c` gibi, SQL'in kendisinden ayrı, psql'e özgü bir komut (`\` ile başlayan).
- **JDBC URL**: hangi driver'a, host'a, porta, ve veritabanına bağlanılacağını kodlayan bir bağlantı string'i (`jdbc:postgresql://host:port/database`).
- **DataSource**: `spring.datasource.*` özelliklerinden auto-configure edilen, uygulama kodunun (Hibernate ve Spring Data JPA üzerinden) PostgreSQL'e ulaşmak için gerçekte kullandığı Spring bean'i.