Databases, Schemas, Tables, and Basic SQL Syntax

Sunucu → veritabanı → şema → tablo hiyerarşisi, CREATE TABLE sözdizimi bu projenin kendi gerçek V1__init_schema.sql'i üzerinden okunarak, kolon tanımları/kısıtlar, SQL yorumları, ve DDL ile DML arasındaki ilk ayrım. PostgreSQL Foundations kategorisinin 3.'sü.

Başlangıç 20 dk

Son güncelleme: 25 Ağustos 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/databases-schemas-tables-and-basic-sql

---

"PostgreSQL'e Bağlanmak" dersi, bu projenin kendi `learning` veritabanına işaret eden gerçek bir `psql` oturumu açtı. Bu ders, o veritabanının içinde gerçekte ne olduğuna bakıyor -- Hibernate üzerinden değil, bu ders için uydurulmuş bir `CREATE TABLE` üzerinden değil, bu projenin kendi gerçek, çalışan `V1__init_schema.sql` migration'ı üzerinden, ilk kez SQL olarak okunarak.

Veritabanı, Şema, ve Tablo Hiyerarşisi

Tek bir PostgreSQL sunucusu birçok **veritabanı** barındırabilir -- "PostgreSQL'e Bağlanmak" dersi bunlardan ikisini ("learning" ve "learning_test") zaten yan yana görmüştü, birbirinden tamamen izole. Bir veritabanının içinde tablolar düz değildir -- bir **şema** içinde yaşarlar, o veritabanı içindeki tabloların (ve başka nesnelerin) adlandırılmış bir ad alanı. Bir veritabanının birden çok şeması olabilir; bir şemanın birçok tablosu olabilir. Tam hiyerarşi şudur: **sunucu → veritabanı → şema → tablo**.

Bu proje hiçbir zaman kendi şemasını oluşturmaz -- `V1__init_schema.sql` dahil her migration'daki her `CREATE TABLE`, örtük olarak `public` adlı şemaya düşer, PostgreSQL'in her yeni veritabanında otomatik oluşturduğu tek şema. "PostgreSQL'e Bağlanmak" dersindeki `\dt`'nin `topic` / `category` / `course`'u hiç bir şema adından bahsetmeden listelemesinin nedeni tam olarak budur -- `\dt` varsayılan olarak `public` şemasını gösterir, ve bu proje için `public` şimdiye kadar önemli olan tek şemadır.

```
learning=# \dn
  List of schemas
  Name  |  Owner
--------+----------
 public | learning
```

`\dn` -- henüz görülmemiş bir `psql` meta-komutu -- mevcut veritabanındaki her şemayı listeler. Bu proje gibi çoğu tek-uygulamalı proje için, bu listede tek bir satır olur.

CREATE TABLE: Bu Projenin Kendi Gerçek Şemasını Okumak

Bu projenin ilk migration'ı `V1__init_schema.sql`, gerçek, değiştirilmemiş SQL'dir -- Flyway'in bu projenin sahip olduğu her ortama, ilk commit'ten bugüne kadar çalıştırdığı aynı dosya:

```
CREATE TABLE course
(
    id   BIGSERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    slug VARCHAR(255) NOT NULL UNIQUE
);

CREATE TABLE category
(
    id        BIGSERIAL PRIMARY KEY,
    course_id BIGINT       NOT NULL REFERENCES course (id) ON DELETE CASCADE,
    name      VARCHAR(255) NOT NULL,
    slug      VARCHAR(255) NOT NULL,
    CONSTRAINT uq_category_course_slug UNIQUE (course_id, slug)
);
```

`CREATE TABLE <ad> ( <kolon tanımları>, <tablo-seviyesi kısıtlar> );` -- ifadenin tüm şekli budur: bir ad, sonra parantez içinde, virgülle ayrılmış bir liste. Burada bu ders için uydurulmuş hiçbir şey yok: bu, Spring Data JPA'nın `Course` entity'sinin eşlendiği tam olarak o tablo, ve `psql`'in `\d course` komutunun ("PostgreSQL'e Bağlanmak"tan) tarif edeceği tam olarak o tablo.

Kolon Tanımları: Tür, Kısıtlar, Varsayılanlar

Parantez içindeki her satır bir kolondur: bir ad, bir veri türü, sonra soldan sağa okunan sıfır veya daha fazla kısıt.

`id BIGSERIAL PRIMARY KEY` -- `BIGSERIAL`, artan bir 64-bit tamsayıyı otomatik üreten pratik bir PostgreSQL türüdür (tam veri türü kapsamı, bu projenin özellikle neden `BIGSERIAL` kullandığı dahil, bir sonraki dersin işi); `PRIMARY KEY`, bu kolonu satırın benzersiz kimliği olarak işaretler. `name VARCHAR(255) NOT NULL` -- 255 karakterle sınırlı, asla `NULL` olamayan değişken uzunlukta bir string. `slug VARCHAR(255) NOT NULL UNIQUE` -- aynı tür ve not-null kuralı, artı bu tabloda hiçbir iki satırın aynı değeri paylaşamayacağı bir kısıt. `course_id BIGINT NOT NULL REFERENCES course (id) ON DELETE CASCADE` -- bir foreign key: `course_id`, `course` tablosundaki mevcut bir `id`ye eşleşmelidir, ve `ON DELETE CASCADE`, bir `course` satırının silinmesinin ona referans veren her `category` satırını otomatik olarak sildiğini söyler. `CONSTRAINT uq_category_course_slug UNIQUE (course_id, slug)`, kolon-seviyesi değil tablo-seviyesi bir kısıttır (birden fazla kolona yayılan) -- bir `slug`ın yalnızca belirli bir `course_id` İÇİNDE benzersiz olması gerektiğini söyler, tüm tablo genelinde değil. Tam kısıt mekaniği --

`PRIMARY KEY` ve `FOREIGN KEY`nin kavram olarak neden var olduğu, `ON DELETE CASCADE`nin alternatiflerinin ne olduğu -- iki ders ileride "Constraints and Keys"e ait; bu dersin işi yalnızca sözdizimini her parçayı tanıyacak kadar akıcı okumak.

İfade Sonlandırıcıları, Büyük/Küçük Harf Duyarlılığı, ve Identifier'lar

Bu projenin migration'larındaki her SQL ifadesi bir noktalı virgülle (`;`) biter -- PostgreSQL, ifadeleri bir yeni satıra kadar değil, o karaktere kadar okur, tek bir `CREATE TABLE`nın güvenle birçok satıra yayılabilmesinin nedeni budur. SQL anahtar kelimeleri (`CREATE TABLE` , `NOT NULL` , `REFERENCES`), bu projenin migration'larının yaptığı gibi geleneksel olarak büyük harfle yazılır, ama PostgreSQL bunu gerçekte zorunlu kılmaz -- `create table` birebir aynı şekilde çalışır. Tırnaksız identifier'lar (tablo ve kolon adları), nasıl yazıldığına bakılmaksızın PostgreSQL tarafından otomatik olarak küçük harfe çevrilir -- bu projenin migration'larının her tablo ve kolon adını zaten küçük harfle yazmasının nedeni tam olarak budur -- PostgreSQL'in zaten yapacağı şeyle savaşmak yerine ona uymak. Bir identifier'ı çift tırnak içine almak (`"Course"`) tam büyük/küçük harfini korur ve o andan itibaren büyük/küçük harf duyarlı yapar -- bu proje bunu hiç yapmaz, ve tırnaklı ile tırnaksız identifier'ları karıştırmak, aşağıdaki Yaygın Hatalar'da işlenen kafa karıştırıcı "table not found" hatalarının yaygın bir kaynağıdır.

SQL'de Yorumlar

`V1__init_schema.sql` , `topic_translation`'ın hemen üstünde iki satırlık bir SQL yorumuna sahip:

```
-- NOT: "published" burada YOK, kasıtlı olarak topic_translation seviyesinde —
-- bir dilin yayında, diğerinin taslak olabilmesi için.
```

`--` , o satırın sonuna kadar giden tek satırlık bir SQL yorumu başlatır -- Java'nın `//` siyle ilgisiz, ama birebir aynı amaca hizmet eder. SQL, blok yorumlarını da destekler (`/* ... */`), gerçi bu projenin migration'ları yalnızca `--` kullanır. Bu özel yorum, CLAUDE.md'nin kendisinin `topic_translation`'ın tasarımı hakkında söylediği tam olarak aynı gerçeği kaydediyor -- bariz olmayan bir şema kararını, onu yapan SQL'in hemen yanında belgelemenin gerçek bir örneği.

DDL vs. DML: İlk Bir Ayrım

`CREATE TABLE` , **DDL** (Data Definition Language / Veri Tanımlama Dili) adı verilen bir SQL kategorisine aittir -- bir veritabanının *yapısını* tanımlayan ya da değiştiren ifadeler: `CREATE TABLE` , `ALTER TABLE` , `DROP TABLE` . `V1__init_schema.sql` dahil, bu projenin her Flyway migration'ı DDL'dir. Ayrı bir kategori olan **DML** (Data Manipulation Language / Veri İşleme Dili) -- `INSERT` , `UPDATE` , `DELETE` , `SELECT` -- DDL'in zaten oluşturduğu bir yapı içindeki *satırları*

okur ve yazar. Bu projenin migration'ları bazen ikisini karıştırır: `V1__init_schema.sql` saf DDL'ken, `V402__connecting_to_postgresql_topic.sql` (bu kursun kendi topic'lerini oluşturan) gibi bir migration saf DML'dir, DDL'in `V1`'de zaten tanımladığı bir `topic` tablosuna satır `INSERT` eder. Bu kategorideki sonraki dört ders -- ekleme/güncelleme/silme, sonra `SELECT`, sonra sıralama/pagination, sonra join'ler -- tamamen DML'dir; bu, "Constraints and Keys"e kadar DDL odaklı son ders.

Yaygın Yanlış Anlamalar

"Bir veritabanı ile bir şema aynı şeydir." Değildir -- veritabanı, `psql`'in `\l`'sinin listelediği ve `\c`'sinin arasında geçiş yaptığı üst-seviye kaptır; şema, bir veritabanının *içindeki* bir ad alanıdır, `\dn` tarafından listelenir. Bu proje her veritabanı için tam olarak bir şemaya (`public`) sahip olduğu için bu ayrımı kaçırmak kolay, ama tek bir veritabanı birçok şema barındırabilir. **"SQL anahtar kelimeleri büyük harf olmak zorundadır."** Zorunda değildir -- PostgreSQL anahtar kelimeler için büyük/küçük harf duyarsızdır; büyük harf, yaygın olarak izlenen saf bir okunabilirlik konvansiyonudur, bu projenin kendi migration'larının da tutarlı şekilde izlediği. **"Tablo ve kolon adları, Java identifier'ları gibi büyük/küçük harf duyarlıdır."** Yalnızca tırnaklanmışsa -- tırnaksız identifier'lar otomatik olarak küçük harfe katlanır, bu yüzden `Course`, `COURSE`, ve `course` -- biri çift tırnakla oluşturulmadığı sürece -- hepsi birebir aynı tabloya işaret eder.

Best Practices

- Bir `CREATE TABLE` ifadesini yukarıdan aşağıya bağımsız kolon tanımlarının bir listesi olarak oku, sonra ayrı olarak sonda tablo-seviyesi `CONSTRAINT` satırlarını ara -- ikisini tek geçişte ayrıştırmaya çalışmak, gerçek bir migration dosyasını olduğundan daha zor hissettiren şeydir.
- Bu projenin migration'larının yaptığı gibi küçük harfli, tırnaksız identifier'lara sadık kal -- bu, bir adın kullanıldığı her yerde tırnaklama disiplini gerektirmek yerine büyük/küçük harf duyarlılığı kafa karışıklığını tamamen atlar.
- Amacı yalnızca adından belli olmayan bir kolon ya da kısıtın hemen üstüne bir SQL yorumu (`--`) yaz -- `V1__init_schema.sql`'in kendi `topic_translation` üstündeki yorumu, taklit edilmeye değer gerçek bir örnek.
- Yabancı bir tabloya ilk kez bakarken, migration dosyasını okumadan önce `psql`'in `\d <table>`sine ("PostgreSQL'e Bağlanmak"tan) başvur -- tablonun *mevcut* yapısını, sonraki her migration onu potansiyel olarak değiştirdikten sonraki hâliyle gösterir, tek başına eski bir `CREATE TABLE` ifadesinin gösteremeyeceği bir şey.

Yaygın Hatalar

- Bir `NOT NULL` kolonunun bir şekilde yinelenen değerleri de engellediğini, ya da `UNIQUE`in bir şekilde `NULL`ı da engellediğini varsaymak -- bunlar bağımsız kısıtlardır; bir `UNIQUE` kolon hâlâ birden fazla `NULL` tutabilir (PostgreSQL, `NULL`ı -- benzersizlik dahil -- başka bir `NULL`a hiçbir zaman eşit saymaz).
- Bir identifier'ı tutarsız şekilde tırnaklamak -- bir tabloyu `"Course"` olarak oluşturup sonra `course`u (tırnaksız, ve bu yüzden küçük harfe katlanmış) sorgulamak, bir yazım-hatası-benzeri ufak bir kaçırma değil, gerçek bir "relation does not exist" hatası üretir.
- `psql`de etkileşimli SQL yazarken bir ifadenin sonundaki noktalı virgülü unutmak -- onsuz, `psql` bir şey çalıştırmak yerine sadece bir devam prompt'unda daha fazla girdi bekler.
- Her SQL ifadesini varsayılan olarak DML sanmak ve bir transaction içindeki bir `CREATE TABLE`nın tam olarak bir `INSERT` gibi geri alınabildiğini görünce şaşırmak -- PostgreSQL, bazı veritabanlarının aksine, transactional DDL'i destekler, bu kursta daha sonra "Transactions and Concurrency in PostgreSQL"ın geri döneceği bir ayrıntı.

Özet, Cheat Sheet ve Terimler Sözlüğü

Özet

- Tam hiyerarşi sunucu → veritabanı → şema → tablodur; bu projenin her veritabanı için bir şeması (`public`) var, `\dn` tarafından listelenen.
- `CREATE TABLE <ad> ( <kolonlar>, <kısıtlar> );` DDL'dir -- satırları değil yapıyı tanımlar; `V1__init_schema.sql`, bu projede hâlihazırda çalışan gerçek DDL'dir.
- Her kolon tanımı soldan sağa okunur: ad, veri türü, sonra kısıtlar (`NOT NULL`, `UNIQUE`, `PRIMARY KEY`, `REFERENCES ... ON DELETE ...`) -- tablo-seviyesi kısıtlar (bileşik bir `UNIQUE` gibi) kolonlardan sonra ayrı olarak listelenir.
- SQL anahtar-kelime-büyük/küçük-harf-duyarsızdır ve tırnaksız identifier'ları küçük harfe katlar; `--` tek satırlık bir yorum başlatır.
- DDL (`CREATE` / `ALTER` / `DROP`) yapıyı tanımlar; DML (`INSERT` / `UPDATE` / `DELETE` / `SELECT`) onun içindeki satırları okur ve yazar -- sonraki birkaç ders tamamen DML'dir.

Cheat Sheet

```
-- Minimal CREATE TABLE şekli
CREATE TABLE tablo_adi
(
    kolon_adi TÜR kısıt kısıt,
    CONSTRAINT kısıt_adı KISIT_TÜRÜ (kolon, ...)
);

-- Tek satırlık yorum
-- bunun gibi
```

```
\dn            -- mevcut veritabanındaki şemaları listele (psql)
\dt            -- mevcut şemadaki tabloları listele (psql, "PostgreSQL'e Bağlanmak"tan)
\d <table>     -- bir tablonun kolonlarını tarif et (psql, "PostgreSQL'e Bağlanmak"tan)
```

Terimler Sözlüğü

- **Şema**: bir veritabanı içindeki tabloların (ve başka nesnelerin) adlandırılmış bir ad alanı; bu proje yalnızca varsayılan `public` şemasını kullanır.
- **DDL (Data Definition Language)**: yapıyı tanımlayan ya da değiştiren SQL -- `CREATE TABLE`, `ALTER TABLE`, `DROP TABLE`.
- **DML (Data Manipulation Language)**: satırları okuyan ya da yazan SQL -- `INSERT`, `UPDATE`, `DELETE`, `SELECT`.
- **Identifier**: bir tablo ya da kolon adı; tırnaksız identifier'lar küçük harfe katlanır, tırnaklı olanlar (`"Ad"`) tam büyük/küçük harfini korur ve büyük/küçük harf duyarlı hâle gelir.
- **Tablo-seviyesi kısıt**: tek bir kolona değil, kendi satırına konan bir kısıt (bileşik bir `UNIQUE` gibi), çünkü birden fazla kolona yayılır.