# SELECT and Filtering

SELECT'in temel şekli, SELECT * yerine kolonları açıkça adlandırma, karşılaştırma /mantıksal operatörler, LIKE desen eşleştirme, IN/BETWEEN, ve NULL'ın neden = ile değil yalnızca IS NULL ile filtrelenebildiği -- bu projenin kendi gerçek topic/category tablosu üzerinde. PostgreSQL Foundations kategorisinin 7.'si.

**Başlangıç** 20 dk

Son güncelleme: 25 Ağustos 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/select-and-filtering

---

Bu kursta şimdiye kadar kullanılan her `SELECT`, bir `INSERT` ya da `UPDATE`nin içine gömülü, tam olarak bir slug'a göre tam olarak bir id arayan bir subquery'ydi. Bu ders `SELECT`e ana olay olarak bakıyor -- satırları geri okumak, onları filtrelemek, bu projenin kendi gerçek `topic` ve `category` tabloları üzerinde.

## SELECT: Temel Şekil

```
SELECT slug, difficulty, estimated_minutes
FROM topic
WHERE category_id = 1;
```

`SELECT <kolonlar> FROM <tablo> WHERE <koşul>;` -- önce kolonlar, sonra tablo, sonra isteğe bağlı bir filtre. Bu projenin kendi gerçek verisine karşı çalıştırıldığında, bu ifade `id`si 1 olan kategoriye ait her `topic` satırını, yalnızca üç kolonunu göstererek döndürür.

## Belirli Kolonları Seçmek vs. SELECT *

`SELECT *`, bir tablonun sahip olduğu her kolonu, mevcut kolon sırasıyla döndürür -- keşif yaparken bir `psql` prompt'unda kullanışlı, ama kalıcı olacak herhangi bir şeyde kaçınmaya değer: bir migration bir kolon eklediği anda (bu projenin Flyway migration'larının "Databases, Schemas, Tables, and Basic SQL Syntax" ve "PostgreSQL Data Types"in zaten gösterdiği gibi tabloları sürekli değiştirdiği bir projede gerçek bir olasılık) sessizce şeklini değiştirir, ve akışta hiçbir şeyin ihtiyaç duymadığı kolonları da çeker. Bu dersteki diğer her örneğin yaptığı gibi kolonları açıkça adlandırmak, bir sorgunun çıktı şeklini, tablo daha sonra ne olursa olsun sabit tutar.

## WHERE ve Karşılaştırma Operatörleri

PostgreSQL'in karşılaştırma operatörleri, herhangi bir programlama dilinden beklediğinizle aynı, bir yazım farkıyla: eşitlik için `=` (`==` değil), artı "eşit değil" için `<>` ya da `!=` (ikisi de birebir aynı şekilde çalışır), `<`, `>`, `<=`, `>=`.

```
SELECT slug FROM topic WHERE difficulty = 'ADVANCED';
SELECT slug FROM topic WHERE estimated_minutes > 20;
SELECT slug FROM course WHERE slug <> 'postgresql';
```

İlki, bu projenin kendi verisine karşı gerçek -- `difficulty` metin olarak saklanır (`Difficulty`, "PostgreSQL Data Types"in SQL açısından düz bir `VARCHAR` kolonu olarak sınıflandıracağı `@Enumerated(EnumType.STRING)` ile eşlenir), bu yüzden herhangi bir string kolonun karşılaştırılacağı şekilde, literal'in etrafında tek tırnaklarla karşılaştırılır.

## Mantıksal Operatörler: AND, OR, NOT

Birden fazla koşul, tam olarak Java'nın `&&`, `||`, ve `!`si gibi `AND`, `OR`, ve `NOT` ile birleşir:

```
SELECT slug FROM topic
WHERE difficulty = 'ADVANCED' AND estimated_minutes > 25;

SELECT slug FROM topic
WHERE difficulty = 'BEGINNER' OR difficulty = 'INTERMEDIATE';
```

`AND`, `OR`den daha sıkı bağlanır, aritmetiğin `*`sinin `+`ya göre sahip olduğu aynı öncelik sırası -- bunları parantez olmadan karıştırmak, bir filtrenin amaçlanandan daha fazla ya da daha az satırla sessizce eşleşmesinin gerçek, yaygın bir kaynağıdır:

```
-- Neredeyse kesinlikle amaçlanan değil: AND, OR'dan önce bağlanır,
-- bu yüzden bu "difficulty = 'ADVANCED' AND category_id = 5"
-- YA DA "difficulty = 'BEGINNER'" (herhangi bir kategori) olarak okunur
WHERE difficulty = 'ADVANCED' OR difficulty = 'BEGINNER' AND category_id = 5

-- Muhtemelen amaçlanan -- parantezler gruplamayı açıkça yapar
WHERE (difficulty = 'ADVANCED' OR difficulty = 'BEGINNER') AND category_id = 5
```

## LIKE ve Desen Eşleştirme

`LIKE`, iki wildcard kullanarak string'leri bir desene karşı eşleştirir: `%` herhangi bir karakter dizisiyle (hiçbiri dahil) eşleşir, `_` tam olarak bir karakterle eşleşir.

```
SELECT slug FROM topic WHERE slug LIKE 'postgresql%';
SELECT slug FROM topic WHERE slug LIKE '%data-types';
SELECT slug FROM topic WHERE slug LIKE '%postgresql%';
```

Bu projenin kendi gerçek `topic` tablosuna karşı çalıştırılan ilki, `postgresql-and-the-relational-model` ve `postgresql-data-types` gibi slug'larla eşleşir -- `connecting-to-postgresql` onunla eşleş*mez* (`%` sonda, bu yüzden yalnızca `postgresql` ile başlayan slug'larla eşleşir), ama bunun yerine `WHERE slug LIKE '%postgresql%'` ile eşleşirdi, her iki tarafta wildcard'larla. `LIKE`, varsayılan olarak büyük/küçük harf duyarlıdır; `ILIKE`, büyük/küçük harf duyarsız versiyondur, PostgreSQL'e özgüdür (standart SQL'in parçası değil).

## IN ve BETWEEN

`IN`, birkaç `OR`u zincirlemek yerine, tek bir koşulda bir değer listesine karşı eşleştirir:

```
SELECT slug FROM topic
WHERE difficulty IN ('INTERMEDIATE', 'ADVANCED');
```

`difficulty = 'INTERMEDIATE' OR difficulty = 'ADVANCED'` e eşdeğer, ama daha kısa ve daha net -- "Inserting, Updating, and Deleting Data" bu tam operatörü gerçek bir `DELETE ... WHERE example_name IN (...)` de zaten kullanmıştı. `BETWEEN`, tek bir koşulda kapsayıcı bir aralığı kontrol eder:

```
SELECT slug FROM topic
WHERE estimated_minutes BETWEEN 15 AND 20;
```

`estimated_minutes >= 15 AND estimated_minutes <= 20` e eşdeğer -- her iki sınır da dahil, ki bu her dilin range yardımcısı her iki uçta da kapsayıcı olmadığı için unutulması kolay bir şey.

## NULL ve Filtreleme: = NULL Neden Çalışmaz

"Constraints and Keys" zaten PostgreSQL'in bir `NULL`ı asla başka bir `NULL`a eşit saymadığını kurdu -- aynı kural filtrelemeye de uygulanır, ve bir `UNIQUE` kısıtını tökezlettiğinden çok daha sık `WHERE`i tökezletir. Bu projenin kendi `topic.estimated_minutes` kolonu null-olabilir (`slug` ya da `difficulty`nin aksine), bu yüzden bunun önemli olduğu gerçek bir kolon:

```
-- HER ZAMAN SIFIR satır döndürür -- "estimated_minutes'in NULL olduğu satırlar" değil
SELECT slug FROM topic WHERE estimated_minutes = NULL;

-- NULL'ı kontrol etmenin doğru yolu
SELECT slug FROM topic WHERE estimated_minutes IS NULL;

-- Ve tersi
SELECT slug FROM topic WHERE estimated_minutes IS NOT NULL;
```

`= NULL`, tam olarak false değildir -- PostgreSQL'in üç-değerli mantığında (`true` / `false` / `unknown`), herhangi bir şeyi `NULL`la `=` ile karşılaştırmak `unknown`a değerlenir, ve `WHERE`

yalnızca koşulun `true` olduğu satırları tutar, bu yüzden `unknown` bir satır her hâlükârda sessizce düşer. `IS NULL` / `IS NOT NULL`, `NULL`ı test etmenin tek doğru yoludur -- bunlar `=` / `<>` 'den ayrı bir sözdizimi parçasıdır, onların özel bir durumu değil.

## Yaygın Yanlış Anlamalar

**"`SELECT *`, `psql` için zararsız bir kısayoldur."** Anlık keşif için sorun değil, ama onu uygulama koduna ya da kaydedilmiş bir sorguya taşımak, daha sonraki bir `ALTER TABLE ADD COLUMN`ın, migration'ın gerçekleştiği yerden çoğu zaman uzakta, o sorgunun sonuç şeklini sessizce değiştirmesi anlamına gelir. **"`!=` ve `<>` farklı şeyler ifade eder."** Etmezler -- ikisi de "eşit değil"dir, `<>` SQL-standardı yazımıdır ve `!=` yaygın desteklenen bir takma addır; bu projenin kendi stilinin ikisi arasında güçlü bir tercihi yok. **"`LIKE '%text%'` ve full-text search aynı şeydir."** Değildirler -- `LIKE`, kelime sınırları, alaka sıralaması, ya da kök bulma kavramı olmadan literal bir alt-string/desen eşleşmesi yapar; PostgreSQL'in gerçek full-text search'ü (ayrı, daha ileri bir özellik) bu kursun kapsamı dışında.

## Best Practices

- Tek seferlik bir `psql` kontrolünün ötesindeki her şeyde `SELECT *` yerine kolonları açıkça adlandır -- bu, bu projenin kendi migration'ları zamanla kolonlar eklemeye, yeniden adlandırmaya, ve değiştirmeye devam ederken bir sorgunun şeklini sabit tutar.
- Aynı kolonu farklı literal'lerle karşılaştıran bir `OR` zinciri yerine `IN (...)`e başvur -- "Inserting, Updating, and Deleting Data"nın gerçek `DELETE ... WHERE example_name IN (...)`i, bunun hem daha kısa hem daha net olduğunun gerçek bir örneği.
- Varsayılan önceliğin teknik olarak amaçlanan sonucu üretecek olduğu durumlarda bile, ikisi de aynı `WHERE` cümlesinde göründüğünde `AND` / `OR` kombinasyonlarını açıkça parantezle -- bu, okuyucunun hangi operatörün daha sıkı bağlandığını hatırlama ihtiyacını tamamen ortadan kaldırır.
- Null-olabilir herhangi bir kolonu (bu projede `estimated_minutes`), `NULL`a karşı `=` / `<>` ile değil, `IS NULL` / `IS NOT NULL` ile kontrol et -- "hiçbir satır bulunamadı" ile "sessizce yanlış bir sorgu" arasındaki fark çoğu zaman tam olarak buna dayanır.

## Yaygın Hatalar

- `WHERE column = NULL` (ya da `<> NULL`) yazıp null-olabilirlik üzerinde filtreleyeceğini beklemek, ve sorgu hiçbir şey döndürmediğinde ya da eşleşmesi gereken satırları beklenmedik şekilde dışarıda bıraktığında kafa karışıklığı yaşamak.
- `LIKE`nin wildcard konumunun yönsel olarak önemli olduğunu unutmak -- `'postgresql%'` (ile başlar) ve `'%postgresql'` (ile biter), gerçekten farklı satırlar döndürür, ve hiçbiri her iki uçta da wildcard olmadan genel bir "içerir" araması değildir.

- Bir `WHERE` cümlesinde `AND` ve `OR`u parantez olmadan karıştırmak ve operatör önceliğinin "açıkça" amaçlananı ifade edeceğine güvenmek -- genellikle hatasız derlenir ve çalışır, ki bu tam olarak ortaya çıkan yanlış satır sayısını fark etmeyi kolayca kaçırılır kılan şeydir.
- `IN (...)`in, değiştirdiği eşdeğer `OR` zincirinin yaptığından daha fazlasını yaptığını sanmak -- her değer için tek bir `=` karşılaştırmasının zaten sağlayacağının ötesinde otomatik yineleme-giderme ya da tür dönüşümü sunmaz.

## Özet, Cheat Sheet ve Terimler Sözlüğü

### Özet

- `SELECT <kolonlar> FROM <tablo> WHERE <koşul>;` tam şekildir; `SELECT *` yerine kolonları açıkça adlandırmak, bu projenin şeması geliştikçe bir sorgunun sonuç şeklini sabit tutar.
- Karşılaştırma operatörleri (`=`, `<>`/`!=`, `<`, `>`, `<=`, `>=`) beklendiği gibi çalışır; `AND`/`OR`/`NOT` koşulları birleştirir, `AND` `OR`dan daha sıkı bağlanır -- bunları karıştırırken açıkça parantezle.
- `LIKE`, `%` (herhangi bir dizi) ve `_` (tam olarak bir karakter) ile bir deseni eşleştirir, varsayılan olarak büyük/küçük harf duyarlı; `ILIKE`, PostgreSQL'in büyük/küçük harf duyarsız versiyonudur.
- `IN (...)`, aynı kolon üzerindeki bir `OR` zincirinin yerini alır; `BETWEEN a AND b`, kapsayıcı bir aralık kontrolüdür.
- `NULL`, `=`/`<>` altında hiçbir şeye -- başka bir `NULL` dahil -- asla eşit değildir -- `IS NULL`/`IS NOT NULL`, onu filtrelemenin tek doğru yoludur, "Constraints and Keys"in `UNIQUE` için zaten kurduğu aynı kuralın doğrudan bir devamı.

### Cheat Sheet

```
SELECT a, b FROM t WHERE a = 1;
SELECT a, b FROM t WHERE a <> 1 AND b > 5;
SELECT a, b FROM t WHERE a LIKE 'foo%';
SELECT a, b FROM t WHERE a IN (1, 2, 3);
SELECT a, b FROM t WHERE a BETWEEN 1 AND 10;
SELECT a, b FROM t WHERE a IS NULL;
SELECT a, b FROM t WHERE a IS NOT NULL;
```

### Terimler Sözlüğü

- **Wildcard**: bir `LIKE` deseninde özel bir karakter -- `%` herhangi bir karakter dizisi için, `_` tam olarak biri için.
- **Üç-değerli mantık**: PostgreSQL'in `true`/`false`/`unknown` değerlendirme modeli, `= NULL` ın hiçbir şeyle asla eşleşmemesinin nedeni.
- **ILIKE**: `LIKE`nin PostgreSQL'e özgü, büyük/küçük harf duyarsız eşdeğeri, standart SQL'in parçası değil.
- **Predicate**: verilen bir satır için true, false, ya da unknown'a değerlenen bir `WHERE` cümlesindeki herhangi bir koşul için genel terim.