Inserting, Updating, and Deleting Data

INSERT/UPDATE/DELETE, bu projenin kendi gerçek INSERT ... SELECT ve UPDATE ... WHERE migration'larıyla, RETURNING, ON CONFLICT (upsert), ve Flyway'in migration'ların neden hiç ON CONFLICT'e ihtiyaç duymadığını nasıl garanti ettiği. PostgreSQL Foundations kategorisinin 6.'sı.

Orta 25 dk

Son güncelleme: 25 Ağustos 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/inserting-updating-and-deleting-data

---

Bu kurstaki her ders şimdiye kadar salt-okunurdu -- yapıya (`CREATE TABLE`) ve türlere bakıldı, ama hiçbir satır gerçekte değiştirilmedi. Bu ders bunun değiştiği yer: `INSERT`, `UPDATE`, ve `DELETE`, bu projenin kendi gerçek migration'ları üzerinden okunuyor -- ki bu migration'lar `V1` den beri satırları tam olarak bu şekilde yazıyor.

INSERT: Temel Şekil

En basit form, tabloyu, kolonları, ve değerleri adlandırır:

```
INSERT INTO course (name, slug, sort_order)
VALUES ('PostgreSQL', 'postgresql', 5);
```

Bu gerçek -- bu kursun kendi `course` satırını oluşturan tam ifade. Parantez içindeki kolon sırası, sonrasında gelen değer sırasıyla eşleşmelidir; dışarıda bırakılan herhangi bir kolon (("PostgreSQL Data Types"ta kapsanan `BIGSERIAL` tarafından üretilen `id` gibi) kendi varsayılanını alır.

Bu Projenin Kendi INSERT ... SELECT Deseni

Düz bir `VALUES` listesi yalnızca her değer bir literal olduğunda çalışır. Bu projedeki neredeyse her migration bunun yerine bir foreign key'e -- bir `category_id` ya da `topic_id`ye -- ihtiyaç duyar ve o üretilmiş sayıyı önceden bilmez. Bu projenin her yerde kullandığı desen `INSERT ... SELECT`tir, id'yi ekleme anında slug'a göre arayarak:

```
INSERT INTO topic (category_id, slug, difficulty, estimated_minutes, sort_order)
SELECT id, 'connecting-to-postgresql', 'BEGINNER', 15, 2
FROM category
WHERE slug = 'postgresql-foundations';
```

Bir `VALUES` cümlesi yerine, kolonlar bir `SELECT`ten doldurulur -- `id`, `WHERE slug = ...`ın eşleştirdiği hangi `category` satırından geliyorsa ondan gelir, ve gerisi hâlâ literaldir. Bu, iki ders ileride "SELECT and Filtering"in düzgünce kapsayacağı aynı `SELECT` sözdizimidir -- şimdilik, "bu sorgunun sonucunu tek bir satır olarak ekle" olarak ele al. Bu, bir migration'ın `V1`den beri her migration'ı çalıştırmış bir veritabanı ile taze bir veritabanı arasında farklı olabilecek belirli bir sayıyı hiç sabit kodlamadan `postgresql-foundations`ın `category_id`sine referans vermesini sağlayan şeydir.

UPDATE: Mevcut Satırları Değiştirmek

`UPDATE`, yeni satırlar oluşturmak yerine mevcut satırları değiştirir, ve -- kritik olarak -- tablodaki her satırı yeniden yazmaktan kaçınmak için bir `WHERE` cümlesine ihtiyaç duyar:

```
UPDATE topic_translation
SET published = true
WHERE language = 'en'
  AND topic_id = (SELECT id FROM topic WHERE slug = 'connecting-to-postgresql');
```

Bu, tam olarak bu kursun ikinci dersini İngilizce'de yayına alan gerçek ifade -- CLAUDE.md'nin proje-geneli bir konvansiyon olarak belgelediği iki-adımlı yayın deseni (Türkçe hemen yayına alınır, İngilizce sonra çevrilir), bundan daha egzotik hiçbir şey üzerinde çalışmaz. `SET`, virgülle ayrılmış birden fazla kolonu aynı anda güncelleyebilir (`SET published = true, seo_title = '...'`), ve `WHERE` cümlesi gerektiği kadar spesifik olabilir -- burada bir subquery (başka bir ifadenin içine gömülü bir `SELECT`), doğru `topic_id`yi slug'a göre bulur, yukarıdaki `INSERT ... SELECT` deseninin kullandığı birebir aynı teknik.

DELETE: Satırları Kaldırmak

`DELETE`, tüm satırları kaldırır, ve `UPDATE` gibi, aynı nedenle bir `WHERE` cümlesine ihtiyaç duyar -- onu dışarıda bırakmak tablodaki her satırı siler:

```
DELETE FROM code_example
WHERE example_name IN ('CardBase', 'CardDemo')
  AND topic_id = (SELECT id FROM topic WHERE slug = 'component-composition');
```

Bu, bir içerik değişikliğinden sonra artık o dersin markdown'ında embed edilmeyen iki örnek satırı kaldıran, React kursundan gerçek bir migration -- `DELETE`, `code_example` tablosunun yapısına hiç dokunmaz, yalnızca bu koşula uyan iki satırı. Buradaki `IN (...)`, `=` ile tek bir tam değer yerine bir değer listesinden herhangi biriyle eşleşir -- "SELECT and Filtering"in bunu düzgünce kapsamasından önce şimdi tanınmaya değer.

RETURNING: Bir Yazma İşleminden Bir Satır Geri Almak

Bir `INSERT`, `UPDATE`, ya da `DELETE`, normalde yalnızca kaç satırın etkilendiğini raporlar -- `RETURNING`, sonradan ayrı bir `SELECT` olmadan, aynı ifadede, gerçek satır verisini geri vermesini sağlar:

```
INSERT INTO course (name, slug, sort_order)
VALUES ('PostgreSQL', 'postgresql', 5)
RETURNING id;
```

Bu, PostgreSQL'in `BIGSERIAL` üzerinden az önce ürettiği yeni `id`yi hemen döndürür -- uygulama kodunun o üretilmiş değere onu sorgulamak için ayrı bir gidiş-dönüş olmadan hemen ihtiyaç duyduğu her yerde kullanışlı. `RETURNING`, `UPDATE` ve `DELETE` üzerinde de birebir aynı şekilde çalışır, satırı güncellemeden *sonraki* ya da silmeden *önceki* hâliyle döndürür. Bu projenin kendi migration'ları bunu hiç kullanmaz -- bir Flyway migration'ı bir kez, gözetimsiz çalışır ve geri bir değer almayı bekleyen çağıran bir kod yoktur -- ama Hibernate'in bir repository'nin `save(...)`i aracılığıyla az önce eklediği bir satırın veritabanı tarafından üretilen `id`sine ihtiyaç duyduğu her yerde içeride güvendiği tam olarak budur.

INSERT ... ON CONFLICT: Upsert

`ON CONFLICT`, bir satır bir `UNIQUE` ya da `PRIMARY KEY` kısıtını ihlal ettiğinde başarısız olmak yerine `INSERT`e ne yapması gerektiğini söyler -- "Constraints and Keys"in zaten kapsadığı kısıt mekaniği, `ON CONFLICT`in tepki verdiği tam olarak şeydir:

```
INSERT INTO category (course_id, name, slug, sort_order)
VALUES (5, 'PostgreSQL Foundations', 'postgresql-foundations', 1)
ON CONFLICT (course_id, slug) DO NOTHING;
```

`ON CONFLICT (course_id, slug)`, izlenen tam kısıtı adlandırır -- burada, bu projenin "Constraints and Keys"teki kendi gerçek `uq_category_course_slug` bileşik `UNIQUE`si. `DO NOTHING`, eşleşen bir satır zaten varsa, bir kısıt-ihlali hatası yükseltmek yerine ekleme işlemini sessizce atlar. Alternatif, `DO UPDATE`, çakışan satırı atlamak yerine günceller:

```
INSERT INTO category (course_id, name, slug, sort_order)
VALUES (5, 'PostgreSQL Foundations', 'postgresql-foundations', 1)
ON CONFLICT (course_id, slug)
DO UPDATE SET name = EXCLUDED.name;
```

`EXCLUDED`, eklenmek üzere olan satıra işaret eder -- `EXCLUDED.name`, yeni `'PostgreSQL Foundations'` değeridir, onu `category.name`den -- *mevcut* satırın şu anki değerinden -- ayırt eder. Bu birleşik "ekle, ya da zaten oradaysa güncelle" davranışına yaygın olarak **upsert** denir.

Bu Projenin Migration'ları Neden Hiç ON CONFLICT Kullanmaz

Bu derste şimdiye kadar gösterilen her `INSERT`, gerçek bir Flyway migration'ından geliyor, ve hiçbiri `ON CONFLICT` kullanmıyor -- yalnızca not etmeye değil, NEDEN olduğunu anlamaya değer. Flyway, her numaralandırılmış migration'ın her veritabanı için tam olarak bir kez, sırayla çalışacağını garanti eder ve değişikliğe karşı checksum'lanır -- bu yüzden `postgresql`/`postgresql-foundations` ekleyen bir migration, hiçbir satırın henüz var olmadığını güvenle varsayabilir; ele alınacak bir çakışma yoktur çünkü Flyway'in kendisi bir çakışmayı önleyen mekanizmadır. `ON CONFLICT`, birden fazla kez, zaten orada olabilecek veriye karşı çalışabilecek kodda yerini hak eder -- elle yeniden çalıştırılan bir veri-yükleme script'i, idempotent bir seed script'i, ya da (uygulama koduna daha yakın) doğal bir key zaten var olup olmadığına bağlı olarak insert-ya-da-update yapmak istenen bir `save(...)` çağrısı. Bu projenin Java kodu da aynı temel nedenle buna hiç başvurmaz: `JpaRepository.save(...)`, `@Id`sinin `null` olup olmadığına göre insert-vs-update'e zaten karar verir, "Entities and the Repository Abstraction"ın zaten kapsadığı gibi -- bir katman yukarıda ilişkili bir problemi çözen farklı bir mekanizma.

Yaygın Yanlış Anlamalar

"`WHERE`siz `UPDATE`/`DELETE` yalnızca 'mevcut' satırı etkiler." SQL'de böyle bir kavram yoktur -- `WHERE`i dışarıda bırakmak, hiçbir onay istemi olmadan, anında tablodaki her satırı hedefler. **"`RETURNING` ikinci bir sorgu çalıştırır."** Çalıştırmaz -- aynı tek ifadedir, yazma işlemini gerçekleştirirken zaten hesapladığı veriyi döndürür, ekstra bir gidiş-dönüş değil. **"`ON CONFLICT`, `INSERT` sırasındaki herhangi bir hatayı yakalar."** Yalnızca adlandırılan belirli kolon(lar) üzerindeki bir `UNIQUE`/`PRIMARY KEY`/`EXCLUDE` kısıt ihlalini yakalar -- bir `NOT NULL` ihlali ya da bir foreign key ihlali ifadeyi hâlâ doğrudan başarısız kılar.

Best Practices

- Bir `UPDATE` ya da `DELETE`nin `WHERE` cümlesini, `SET`/kolonlardan önce, zihinsel olarak önce yaz -- ve onu değiştirmeye karar vermeden önce hangi satırların tam olarak etkileneceğini görmek için önce birebir aynı koşulla bir `SELECT` olarak test etmeyi düşün.
- Bir foreign key'in sayısal id'sini sabit kodlamak yerine `INSERT ... SELECT`i (bu projenin migration'ları boyunca kendi deseni) tercih et -- belirli bir veritabanında bir `category` ya da `topic`in sonunda hangi id'yi aldığından bağımsız olarak doğru kalır.
- Kodun bir yazma işleminin az önce ürettiği bir değere (üretilmiş bir id, hesaplanmış bir varsayılan) ihtiyaç duyduğu her yerde takip eden bir `SELECT` yerine `RETURNING`e başvur -- iki yerine bir gidiş-dönüştür, ve yazma işleminin kendisiyle atomiktir.
- `ON CONFLICT`i varsayılan bir alışkanlık değil bir sinyal olarak ele al -- ona ihtiyaç duymak genellikle aynı işlemin aynı veriye karşı birden fazla kez çalışabileceği anlamına gelir, bu da refleksif olarak başvurmak yerine açıkça adlandırılmaya değer (idempotent bir script, doğal-key bir upsert).

Yaygın Hatalar

- "Yalnızca bu bir satırı" test etme niyetiyle `psql`de bir `WHERE` cümlesi olmadan bir `UPDATE` ya da `DELETE` çalıştırmak -- SQL'de o niyeti zorunlu kılan hiçbir şey yok; ifade çalıştığı anda niteliksiz her satır etkilenir.
- `INSERT ... SELECT`in `SELECT`inin sıfır satır döndürebileceğini (hiçbir şeyle eşleşmeyen bir slug) unutmak -- `INSERT` sonra hatasız şekilde sessizce sıfır satır da ekler, ki bu bir `VALUES` literal'indeki bir yazım hatasından çok daha sessiz bir başarısızlıktır.
- `EXCLUDED`in tabloda zaten var olan mevcut satıra değil, *eklenecek olan* satıra işaret ettiğini fark etmeden `ON CONFLICT DO UPDATE` kullanmak -- `SET name = EXCLUDED.name` yerine `SET name = name` yazmak, eski değeri sessizce sonsuza kadar korur.
- `DELETE FROM table`in (WHERE'siz) `TRUNCATE TABLE table`e eşdeğer olduğunu varsaymak -- ikisi de tabloyu boşaltır, ama `DELETE` bunu satır satır yapar (büyük bir tabloda daha yavaş, ve herhangi bir trigger'ı tetikler) oysa `TRUNCATE` çok daha hızlı yapısal bir işlemdir; tablo boyutları önemsiz olmaktan çıktığında bu ayrım önemli olur.

Özet, Cheat Sheet ve Terimler Sözlüğü

Özet

- `INSERT INTO table (kolonlar) VALUES (...)` yeni bir satır ekler; bu projenin kendi migration'ları neredeyse her zaman bunun yerine `INSERT ... SELECT` kullanır, bir foreign key'in id'sini bir sayıyı sabit kodlamak yerine slug'a göre arayarak.
- `UPDATE table SET kolon = değer WHERE ...` ve `DELETE FROM table WHERE ...`, tüm tabloyu etkilemekten kaçınmak için ikisi de bir `WHERE` cümlesi gerektirir -- bu projenin gerçek yayın-İngilizce migration'ları canlı bir `UPDATE ... WHERE` örneğidir.
- `RETURNING`, ayrı bir takip eden `SELECT`ten kaçınarak, etkilenen satırın verisini aynı ifadede geri verir -- Hibernate, üretilen id'ler için içeride aynı fikre güvenir, gerçi bu projenin kendi migration'ları bunu hiç doğrudan kullanmaz.
- `INSERT ... ON CONFLICT (kolonlar) DO NOTHING / DO UPDATE ...` bir upsert'tir, adlandırılan tam `UNIQUE`/`PRIMARY KEY` kısıtına tepki verir -- bu projenin migration'larının buna hiç ihtiyacı yoktur, çünkü Flyway'in kendisi her migration'ın tam olarak bir kez çalışacağını garanti eder.
- `EXCLUDED`, bir `ON CONFLICT DO UPDATE` cümlesi içinde eklenmek üzere olan satıra işaret eder, tabloda zaten var olan mevcut satırdan ayrı olarak.

Cheat Sheet

```
INSERT INTO t (a, b) VALUES (1, 2);
INSERT INTO t (a, b) SELECT id, 'x' FROM other WHERE slug = 'y';

UPDATE t SET a = 1 WHERE id = 5;
DELETE FROM t WHERE id = 5;

INSERT INTO t (a, b) VALUES (1, 2)
RETURNING id;

INSERT INTO t (a, b) VALUES (1, 2)
ON CONFLICT (a) DO NOTHING;

INSERT INTO t (a, b) VALUES (1, 2)
ON CONFLICT (a) DO UPDATE SET b = EXCLUDED.b;
```

Terimler Sözlüğü

- **Upsert**: yeni bir satır ekleyen, ya da çakışan bir key zaten mevcutsa mevcut satırı güncelleyen bir yazma işlemi -- PostgreSQL'in bunun versiyonu `INSERT ... ON CONFLICT ... DO UPDATE` dir.
- **RETURNING**: `INSERT` / `UPDATE` / `DELETE` üzerinde, etkilenen satırın verisini aynı ifadenin parçası olarak geri veren bir cümle.
- **EXCLUDED**: `ON CONFLICT DO UPDATE` içinde, eklenmek üzere olan satıra işaret eden sözde-tablo.
- **Subquery**: başka bir ifadenin içine gömülü bir `SELECT` -- yukarıda hem `INSERT ... SELECT` te hem bir `WHERE ... = (SELECT ...)` koşulunda kullanıldı.