Constraints and Keys

PRIMARY KEY, FOREIGN KEY ve referential integrity, ON DELETE CASCADE vs. RESTRICT (bu projenin kendi quiz_question_link tablosundaki gerçek kontrastıyla), composite UNIQUE, CHECK kısıtları, ve BIGSERIAL PRIMARY KEY'in GenerationType.IDENTITY'ye nasıl bağlandığı. PostgreSQL Foundations kategorisinin 5.'si.

Başlangıç 20 dk

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

---

"Databases, Schemas, Tables, and Basic SQL Syntax" dersi, `V1__init_schema.sql`'i okurken `PRIMARY KEY`, `REFERENCES ... ON DELETE CASCADE`, `NOT NULL`, ve `UNIQUE`'i zaten kullanmıştı, ama yalnızca tanınacak sözdizimi olarak, tam mekanik açıkça bu derse ertelenerek. Şimdi her biri düzgünce parçalarına ayrılıyor -- gerçekte neyi zorunlu kıldığı, ihlal edildiğinde ne olduğu, ve bu projenin kendi gerçek şemasının pratikte birden fazla seçeneği nerede gösterdiği.

PRIMARY KEY: Gerçekte Neyi Zorunlu Kılar

`PRIMARY KEY`, gerçekte birlikte paketlenmiş iki kısıttır: `NOT NULL` (bir primary key kolonu asla boş olamaz) artı `UNIQUE` (hiçbir iki satır aynı değeri paylaşamaz) -- ve PostgreSQL otomatik olarak üzerine bir index inşa eder, bu kursta daha sonra "Indexes and Query Performance"ın önemini açıkladığı bir şey. Bir tablonun en fazla bir primary key'i olabilir, gerçi bu key birden fazla kolona yayılabilir (bileşik bir primary key) -- bu proje hiçbir yerde bunu kullanmaz, çünkü her tablonun kimliği tek bir üretilmiş `id` kolonudur, ama bir satırın kimliği doğal olarak tek bir üretilmiş sayı yerine değerlerin bir kombinasyonu olduğunda bu seçenek mevcuttur.

FOREIGN KEY ve Referential Integrity

Bir `FOREIGN KEY` (`REFERENCES` ile tanıtılan), bir kolonun değerlerini yalnızca başka bir tabloda zaten bir primary key (ya da unique bir kolon) olarak var olanlarla sınırlar. PostgreSQL'in **referential integrity** dediği şey budur: veritabanı seviyesinde, `category.course_id`'nin `course`'da gerçek bir satıra karşılık gelmeyen bir değer tutması mümkün DEĞİLDİR -- uygulama kodu kontrol ettiği için değil, PostgreSQL'in kendisi `INSERT` ya da `UPDATE`'i doğrudan reddettiği için. Böyle bir kurs yokken bir `category` satırını `course_id`si `9999` olarak eklemeye çalışmak, Java kodunun önceden neyi doğrulayıp doğrulamadığından bağımsız olarak, gerçek, özel bir hatayla (`violates foreign key constraint`) başarısız olur.

ON DELETE Davranışı: CASCADE vs. RESTRICT, Bu Projede Gerçek Bir Kontrast

Tek başına bir foreign key, işaret ettiği satır silindiğinde ne olması gerektiğini söylemez -- bunu `ON DELETE` belirtir, ve bu projenin kendi şeması, bilinçli olarak, iki farklı nedenle iki farklı cevap kullanır.

`V1__init_schema.sql`, içerik hiyerarşisi boyunca `ON DELETE CASCADE` kullanır:

```
category_id BIGINT NOT NULL REFERENCES category (id) ON DELETE CASCADE
```

Bir `category`yi silmek, ona referans veren her `topic`i otomatik olarak siler, bu da her `topic_translation` ve `code_example`a doğru daha ileri cascade olur -- bir satırı silmek kasıtlı olarak bağımlı bir alt ağacın tamamını siler, ki bu burada doğru davranıştır çünkü bir `topic` in kendi `category`si olmadan gerçekten bir anlamı yoktur.

Bu projenin çok daha sonra eklenen `quiz_question_link` tablosu, bilinçli olarak zıt seçimi yapar, nedenini açıklayan kendi gerçek migration yorumuyla:

```
-- question_id: ON DELETE RESTRICT KASITLI -- bir soru, canlı bir sabit quiz'in
-- parçası olduğu sürece hard-delete edilemez.
quiz_id     BIGINT NOT NULL REFERENCES quiz (id) ON DELETE CASCADE,
question_id BIGINT NOT NULL REFERENCES question (id) ON DELETE RESTRICT
```

`ON DELETE RESTRICT`, `CASCADE`'in tam tersini yapar -- silmeyi doğrudan *engeller*: bir `question`, yayınlanmış herhangi bir quiz'e bağlı olduğu sürece, `DELETE FROM question WHERE id = ...` denemek, bağlantıyı onunla birlikte sessizce kaldırmak yerine bir hatayla başarısız olur. Aynı tablonun `quiz_id` için `CASCADE` (bir `Quiz`ı silmek kendi bağlantılarını da götürmeli) ve `question_id` için `RESTRICT` (bir `Question`ı silmek ona bağımlı bir quiz'i sessizce bozmamalı) kullandığına dikkat et -- iki kolon bilinçli olarak farklı davranıyor, çünkü iki ilişki farklı şeyler ifade ediyor. Üçüncü bir seçenek olan `ON DELETE SET NULL`, silmek ya da engellemek yerine foreign key kolonunu `NULL`a ayarlar -- bu projede hiçbir yerde kullanılmaz, çünkü buradaki her foreign key `NOT NULL`dır ve bu yüzden zaten `NULL` tutamaz.

NOT NULL ve UNIQUE (Tek Kolon ve Bileşik)

`NOT NULL` ve `UNIQUE`, "Databases, Schemas, Tables, and Basic SQL Syntax"ta sözdizimi olarak zaten okunmuştu; burada eklemeye değer tek şey, her birinin tek-kolon ve bileşik versiyonu arasındaki ayrım. `course.slug UNIQUE` gibi tek-kolonlu bir `UNIQUE`, tüm tablodaki hiçbir iki satırın o değeri paylaşmasını yasaklar. Bu projenin gerçek

```
CONSTRAINT uq_category_course_slug UNIQUE (course_id, slug)
```

gibi bileşik (tablo-seviyesi) bir `UNIQUE` ise yalnızca her iki kolonun *kombinasyonu* genelinde yinelemeyi yasaklar -- iki farklı kurs, her biri `fundamentals` slug'lı bir kategoriye özgürce sahip olabilir, ama aynı kurs iki tanesine sahip olamaz. `NOT NULL`ın bileşik bir formu yoktur -- her zaman tek seferde bir kolon değerlendirilir -- ama tam olarak burada olduğu gibi çok sık bileşik bir `UNIQUE`le birlikte görünür, çünkü `NULL` olabilecek bir kolon üzerindeki bileşik bir benzersizlik kontrolü aşağıda işlenen `NULL`-ilişkili sürprize takılır.

CHECK Kısıtları

Bir `CHECK` kısıtı, bir kolonun değeri üzerinde keyfi bir boolean koşulu zorunlu kılar, her `INSERT` ve `UPDATE`de değerlendirilir -- örneğin `estimated_minutes INTEGER CHECK (estimated_minutes > 0)`, o değerin sıfır ya da negatif olduğu herhangi bir satırı reddeder. Bu projenin gerçek `topic.estimated_minutes` kolonunun bugün hiçbir `CHECK` kısıtı yok -- bu gerçek, dürüst bir boşluk: veritabanı seviyesinde şu anda hiçbir şey bir migration'ın negatif bir değer eklemesini engellemiyor; yalnızca uygulama-seviyesi özen (ve şimdiye kadar, doğru migration'lar) her satırı geçerli tuttu. Bir `CHECK` kısıtının NE İÇİN olduğunun adil bir illüstrasyonu: şu anda yalnızca konvansiyonla geçerli olan bir varsayımı, PostgreSQL'in kendisinin bir satırın ihlal etmesine izin vermediği bir şeye dönüştürmek.

Kısıtları Açıkça Adlandırmak

`PRIMARY KEY`, `NOT NULL`, ve tek-kolonlu `UNIQUE` / `REFERENCES`in hepsi, hiçbiri verilmezse otomatik oluşturulmuş bir ad alır, bu yüzden bu projenin migration'ları bunları hiç adlandırma zahmetine girmez. Birden fazla kolona yayılan tablo-seviyesi kısıtlar farklıdır -- bu proje bunları her zaman `CONSTRAINT uq_category_course_slug UNIQUE (course_id, slug)`de olduğu gibi açıkça adlandırır -- çünkü adlandırılmamış bileşik bir kısıt, bir hata mesajında ya da daha sonraki bir `ALTER TABLE ... DROP CONSTRAINT`de bilinçli seçilmiş bir addan daha zor tanınan otomatik oluşturulmuş bir ad alır (`category_course_id_slug_key` gibi).

SERIAL/IDENTITY'den GenerationType.IDENTITY'e

"PostgreSQL Data Types" dersi `BIGSERIAL`'ı `BIGINT` artı otomatik artan bir sequence olarak zaten kapsadı, ve "Entities and the Repository Abstraction" Java tarafında `GenerationType.IDENTITY`'yi zaten kapsadı -- ikisi de JDBC sınırının karşı uçlarından anlatılan aynı mekanizmadır, burada tekrarlanacak yeni bir şey yok. `PRIMARY KEY` tam olarak anlaşıldığına göre şimdi eklemeye değer olan, tam resim: `BIGSERIAL PRIMARY KEY`, gerçekte katmanlı üç şeydir -- bir `BIGINT` kolonu, varsayılan değerini üreten bir sequence, ve o değeri hem `NOT NULL` hem benzersiz yapan bir `PRIMARY KEY` kısıtı. Modern bir alternatif sözdizimi olan `GENERATED ALWAYS AS IDENTITY`, biraz farklı semantiklerle aynı işi yapar (varsayılan olarak `INSERT`te açık bir değeri reddeder, oysa `BIGSERIAL` buna izin verir) -- bu proje her yerde `BIGSERIAL` kullanır, ve iki identity stratejisi arasındaki ayrım, UUID primary key'ler devreye girdiğinde daha önemli hâle gelir, bu kursta daha sonra "PostgreSQL-Specific Data Types"ın geri döneceği bir konu.

Yaygın Yanlış Anlamalar

"Bir foreign key otomatik olarak ilişkili satırları siler." Yalnızca `ON DELETE CASCADE` böyle söylüyorsa -- hiç `ON DELETE` cümlesi olmadan varsayılan, etkin olarak `RESTRICT` tir: silme engellenir. **"`UNIQUE` de `NOT NULL` ı ima eder."** Etmez -- bir `UNIQUE` kolon birden fazla `NULL` satır tutabilir, çünkü PostgreSQL bir `NULL` ı -- benzersizlik kontrolleri dahil -- asla başka bir `NULL` a eşit saymaz; `NOT NULL` ve `UNIQUE` in ikisini birden kapsayan tek bir kısıt yerine iki ayrı kısıt olarak yazılmasının tam nedeni budur. **"Bileşik bir `UNIQUE (a, b)` , `a` ve `b` üzerinde iki ayrı `UNIQUE` kısıtıyla aynıdır."** Hiç değil -- bileşik bir kısıt yalnızca *kombinasyonun* bir yinelemesini reddeder; iki ayrı tek-kolonlu kısıt, her biri o kolonun tek başına yinelemesini reddeder, çok daha katı bir kural.

Best Practices

- `ON DELETE CASCADE` i yalnızca bir alt satırın kendi üst satırı olmadan gerçekten bir anlamı olmadığında seç (`category` olmadan `topic` gibi); `RESTRICT` i, bir silmenin sessizce yayılmak yerine bilinçli bir karar zorlaması gerektiğinde seç -- bu projenin kendi `quiz_question_link` tablosu, aynı `CREATE TABLE` de farklı kolonlar için ikisini de doğru şekilde yapıyor.
- Bu projenin kendi `uq_category_course_slug` desenini izleyerek her çok-kolonlu kısıtı açıkça adlandır (`CONSTRAINT ad UNIQUE (...)`) -- bu, gelecekteki bir hata mesajını ya da `DROP CONSTRAINT` ı, otomatik oluşturulmuş bir identifier yerine okunabilir bir şeye dönüştürür.
- Bir kolonun geçerliliği yalnızca kendi değerine bağlı olduğunda (`estimated_minutes` için bir minimum gibi) saf olarak uygulama koduna güvenmek yerine bir `CHECK` kısıtına başvur -- aksi hâlde gelecekteki her doğrudan SQL `INSERT` i -- bir migration'dan gelen dahil -- bunu atlar.
- Bileşik bir `UNIQUE` ile `NOT NULL` ı birlikte gözden geçirilmeye değer bir çift olarak ele al -- bileşik bir unique kısıt içindeki null-olabilir bir kolon, fark edilmesi en az önemli olan tam da o satırlar için benzersizlik kontrolünü sessizce etkisiz kılabilir.

Yaygın Hatalar

- Bir `ON DELETE` cümlesinin yokluğunun "hiçbir şey olmaz" anlamına geldiğini varsaymak -- aslında silmenin kendisinin engellendiği anlamına gelir, bu da ilk denendiğinde -- çoğu zaman geliştirme sırasında değil production'da -- bir kısıt-ihlali hatası olarak yüzeye çıkar.
- Bir değerin var olduğunu garanti etmek için yalnızca `UNIQUE` e güvenmek, sonra o "unique" kolonda aynı anda birçok satırın hepsinin `NULL` olduğunu görünce şaşırmak.
- Başka bir tablonun verisine referans veren bir `CHECK` kısıtı yazmak (`CHECK` koşulları yalnızca mevcut satırı görebilir) -- satırlar-arası ya da tablolar-arası doğrulamanın bunun yerine bir trigger ya da uygulama-seviyesi kontrol gerektirdiğini fark etmek yerine.

- Adlandırılmamış bileşik bir kısıttan gelen otomatik oluşturulmuş bir kısıt adının, çevrilmeden kullanıcıya görünen bir hata mesajına sızmasına izin vermek -- `category_course_id_slug_key` gibi bir ad, şema önünüzde olmadan pek bir şey ifade etmez.

Özet, Cheat Sheet ve Terimler Sözlüğü

Özet

- `PRIMARY KEY`, `NOT NULL` artı `UNIQUE` i paketler ve otomatik bir index ekler; bir tablonun en fazla bir tanesi olur, ama birden fazla kolona yayılabilir.
- `FOREIGN KEY` (`REFERENCES`), uygulama kodundan bağımsız olarak veritabanı seviyesinde referential integrity'yi zorunlu kılar; `ON DELETE CASCADE` / `RESTRICT` / `SET NULL`, bağımlı satırlara ne olacağına karar verir -- bu projenin gerçek `quiz_question_link` tablosu, bilinçli olarak `CASCADE` ve `RESTRICT` i yan yana kullanır.
- `UNIQUE` in tek-kolon ve bileşik (tablo-seviyesi, çok-kolonlu) formları vardır; bileşik bir `UNIQUE`, yalnızca tam kombinasyonun yinelemelerini reddeder, kolonlardan birinin tek başına yinelemesini değil.
- `CHECK`, tek bir satırın kendi değerleri üzerinde keyfi bir koşulu zorunlu kılar -- bu projenin `estimated_minutes` kolonu, şu anda hiç olmayan bir kolonun gerçek bir örneğidir.
- `BIGSERIAL PRIMARY KEY`, üç şeyi katmanlar (bir `BIGINT` kolonu, bir sequence, ve bir `PRIMARY KEY` kısıtı) -- Java tarafında "Entities and the Repository Abstraction"da zaten kapsanan `GenerationType.IDENTITY` yle aynı mekanizma.

Cheat Sheet

```
id         BIGSERIAL PRIMARY KEY,
parent_id  BIGINT NOT NULL REFERENCES parent (id) ON DELETE CASCADE,
child_id   BIGINT NOT NULL REFERENCES child (id) ON DELETE RESTRICT,
slug       VARCHAR(255) NOT NULL UNIQUE,
minutes    INTEGER CHECK (minutes > 0),
CONSTRAINT uq_parent_slug UNIQUE (parent_id, slug)
```

Terimler Sözlüğü

- **Referential integrity**: bir foreign key kolonunun yalnızca başka bir yerde gerçek bir satır olarak var olan değerleri tutmasının veritabanı tarafından zorunlu kılınan garantisi.
- **ON DELETE CASCADE**: referans verilen satırı silmek, ona referans veren her satırı otomatik olarak siler.
- **ON DELETE RESTRICT**: referans verilen satırı silmek, ona referans veren herhangi bir satır varken doğrudan engellenir.
- **Bileşik kısıt**: her kolonu bağımsız olarak değil kombinasyona karşı değerlendirilen, birden fazla kolona yayılan bir `UNIQUE`, `PRIMARY KEY`, ya da `CHECK`.

- **CHECK kısıtı**: PostgreSQL tarafından her insert ve update'te zorunlu kılınan, tek bir satırın kendi değerleri üzerinde bir boolean koşul.