PostgreSQL ve İlişkisel Model

İlişkisel bir veritabanının ne olduğu, PostgreSQL'in özellikle ne olduğu, Spring Boot → Spring Data JPA → Hibernate → SQL → PostgreSQL yığını, ve bu projenin gerçek TopicRepository'sinin bir çağrısının bu yığından nasıl geçtiği -- hiç SQL sözdizimi olmadan, yalnızca zihinsel model. PostgreSQL kursunun ve PostgreSQL Foundations kategorisinin 1.'si.

Başlangıç 15 dk

Son güncelleme: 6 Ekim 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/postgresql-and-the-relational-model

---

Bu, yeni PostgreSQL kursunun ilk dersi, ve "PostgreSQL Temelleri" kategorisinin ilk dersi. Java'yı, Spring Boot'u, ve Spring Data JPA'yı zaten biliyorsun -- "JPA, Hibernate ve Spring Data JPA" hatta JPA (bir spesifikasyon) → Hibernate (bir implementasyon) → Spring Data JPA (bir repository soyutlaması) zihinsel modelini zaten inşa etti. O model, veritabanının kendisinden tam bir katman önce durdu. Bu kurs tam olarak orada başlıyor: henüz SQL sözdiziminde değil, ilişkisel bir veritabanının gerçekte ne olduğunda, ve PostgreSQL'in özellikle ne olduğunda -- bu kursun geri kalanının üzerine asılacağı zihinsel model.

İlişkisel Bir Veritabanı Nedir?

İlişkisel bir veritabanı, veriyi satırlar ve sütunlardan oluşan tablolar olarak saklar, ve -- ona adını veren kısım -- bu tabloların, veriyi çoğaltmak ya da tek bir dev yapının içine yuvalamak yerine, paylaşılan değerler üzerinden birbirine BAŞVURMASINA izin verir. Bir `topic` satırı, kategorisinin adını ve açıklamasını satır içinde tekrarlamaz; ayrı bir `category` tablosundaki bir satırı işaret eden bir `category_id` saklar. "İlişkisel" (relational) tam olarak bunu tarif eder: aile ya da yakınlık gibi daha gevşek bir anlamda ilişkili veri değil, paylaşılan sütun değerleri üzerinden bağlanan, ayrı tablolar olarak organize edilmiş veri.

Neden Var?

Bu model standart hâline gelmeden önce, uygulamalar genelde veriyi tek bir belirli programa uyan herhangi bir şekilde saklardı -- düz bir dosya, özel bir binary format, keyfi derinlikte tek bir kayıt içine yuvalanmış veri. Bu, ikinci bir program aynı veriye ihtiyaç duyana kadar, ya da bir veri parçasının orijinal olarak saklandığı tam şeklin dışında bir şeyle bulunması gerekene kadar, ya da iki veri parçasının ikisi de değiştikçe birbiriyle tutarlı kalması gerekene kadar işe yarardı. İlişkisel model, ve bunu zorlamak için inşa edilmiş bir veritabanı, bunu doğrudan çözer: veri bir kez, daha sonra nasıl sorgulanacağını

varsaymayan bir şekilde saklanır, ve ilişkili parçaları tutarlı tutmaktan -- her bir uygulama değil -- veritabanının kendisi sorumludur.

Tarihçe

İlişkisel modelin kendisi, bir IBM araştırmacısı olan Edgar F. Codd tarafından 1970 tarihli bir makalede önerildi -- o zaman için gerçekten yeni bir fikir, veriyi fiziksel olarak diskte nasıl saklandığından bağımsız olarak, tamamen tablolar ve aralarındaki matematiksel ilişkiler cinsinden tarif ediyordu. PostgreSQL'in kendi tarihi, 1980'lerin ortasında UC Berkeley'de, Michael Stonebraker'ın liderliğinde, açıkça daha önceki bir Berkeley veritabanının (Ingres) halefi olarak POSTGRES adı verilen bir araştırma projesine kadar uzanır -- isim tam olarak "Ingres'ten sonra" anlamına gelir. 1990'ların ortasında SQL desteği ve şimdiki adı PostgreSQL'i kazandı, ve o zamandan beri açık kaynak bir proje olarak geliştiriliyor.

PostgreSQL Özellikle Nedir?

PostgreSQL, ilişkisel bir veritabanının belirli, açık kaynak bir implementasyonudur -- "JPA, Hibernate ve Spring Data JPA"nın JPA ve Hibernate için zaten işlediği aynı ilişki (bir spesifikasyon vs onun implementasyonlarından biri), bir katman aşağıda burada tekrar ortaya çıkıyor: SQL'in kendisi geniş çapta standartlaştırılmış bir dildir, ve PostgreSQL onu implement eden gerçek, çalışan bir yazılım parçasıdır (diğerlerinin yanı sıra -- MySQL, Oracle Database, SQL Server). PostgreSQL'i ayıran şey, ve onu "yalnızca genel olarak SQL" yerine özellikle öğrenmeye değer kılan şey, bu kurs boyunca kademeli olarak işleniyor -- katı standart uyumluluğu, gerçekten genişletilebilir bir tür sistemi (bu kursta ilerideki "PostgreSQL-Specific Data Types: UUID, JSON/JSONB, and Arrays"ta işlenen), ve transaction'lara ve eş zamanlılığa belirli, iyi belgelenmiş bir yaklaşım (bu kursun son dersi "Transactions and Concurrency in PostgreSQL"ta tam olarak işlenen). Şimdilik, önemli gerçek daha basit: PostgreSQL, şu anda, bu platformun kendi verisini gerçekten saklayan ve geri getiren şeydir, bu projenin Java kodunun yaptığı her `TopicRepository` çağrısının altında.

Tablolar, Satırlar ve Sütunlar: Temel Zihinsel Model

Bir tablo, hepsi aynı şekli -- kendi adı ve türü olan aynı sütun kümesini -- paylaşan kayıtların adlandırılmış bir koleksiyonudur. Bir satır, o tablo içindeki tek bir kayıttır -- tek bir belirli topic, tek bir belirli kategori. Bir sütun, o tablodaki her satırın bir değere sahip olduğu (ya da sütun izin veriyorsa, açıkça hiçbir değere sahip olmadığı) adlandırılmış, türlü bir yuvadır.

```
tablo: category
+----+-----------------------+--------------------+
| id | name                  | slug               |
+----+-----------------------+--------------------+
| 1  | Spring Data JPA       | spring-data-jpa    |
| 2  | Advanced Spring       | advanced-spring    |
+----+-----------------------+--------------------+

tablo: topic
+----+-------------+--------------------------------------+
| id | category_id | slug                                 |
+----+-------------+--------------------------------------+
| 1  | 1           | jpa-hibernate-and-spring-data-jpa    |
| 2  | 1           | entities-and-repositories            |
+----+-------------+--------------------------------------+
```

Bu bilinçli olarak bu projenin gerçek şemasından alındı, uydurma bir örnek değil -- `topic`'in `category_id` sütunu, "İlişkisel Bir Veritabanı Nedir?"in tarif ettiği "paylaşılan bir değer üzerinden başka bir tabloya başvurmak" fikrinin tam olarak kendisi, örnek uğruna hiçbir şey uydurulmadan.

PostgreSQL Bir Spring Boot Uygulamasında Nereye Oturuyor

Bir Java metot çağrısı ile gerçekten saklanmış bir satır arasındaki her katman, açıkça adlandırılmaya değer, çünkü her biri sıradakine devrediyor.

```
Spring Boot
     ↓
Spring Data JPA
     ↓
Hibernate
     ↓
   SQL
     ↓
PostgreSQL
     ↓
Tablolar / Index'ler / Kısıtlar / Transaction'lar
```

Spring Boot, üzerindeki parçaları otomatik olarak yapılandırır ("Spring Boot Auto-Configuration ve Properties"te zaten işlenen, "JPA, Hibernate ve Spring Data JPA"da tekrar referans verilen). Spring Data JPA, bildirilen bir interface'ten çalışan bir repository implementasyonu üretir. Hibernate, bir repository çağrısını gerçek SQL'e çevirir. SQL, o SQL ifadesinin yazıldığı dildir. PostgreSQL, o SQL'i alan, çalıştıran, ve bir sonuç döndüren gerçek, çalışan veritabanı sürecidir -- ve bu kursun geri kalanının işlediği her şey (tablolar, index'ler, kısıtlar, transaction'lar), PostgreSQL'in kendisinin o çalıştırmanın altında gerçekte yaptığı şeydir.

Bir Repository Metodundan Bir Veritabanı Satırına

Bunu bu projenin kendi kodunun gerçekten yaptığı bir çağrıyla somutlaştırmak, tüm yığın diyagramının bir kerede yerine oturmasına yardımcı olur.

```
topicRepository.findBySlug("records")
     ↓  (Spring Data JPA metot adından bir sorgu çıkarır --
     ↓   bkz. "Query Methods and JPQL with @Query")
Hibernate bir SQL ifadesi inşa eder
     ↓
SELECT * FROM topic WHERE slug = 'records'
     ↓  (PostgreSQL bu SQL'i alır ve çalıştırır)
PostgreSQL "topic" tablosunu tarar, eşleşen satırı bulur
     ↓
tek satır: {id: 8, category_id: 4, slug: 'records', ...}
     ↓
Hibernate o satırı gerçek bir Topic nesnesine geri eşler
     ↓
topicRepository.findBySlug("records") onu döndürür
```

Burada hiçbir şey yeni bir mekanizma değil -- "JPA, Hibernate ve Spring Data JPA" bu tam akışı, tam olarak bu metot için, zaten baştan sona işledi. BU kursta yeni olan, `SELECT` satırından itibaren olan şey: PostgreSQL'in bu ifadeyi almasını, eşleşen satırı gerçekte nasıl bulacağına karar vermesini (bu kursta ilerideki "Indexes and Query Performance with EXPLAIN"in doğrudan işlediği bir konu), ve onu döndürmesi.

ACID: İlk Bir Bakış

Bu kursun son dersi "Transactions and Concurrency in PostgreSQL"a kadar hiçbiri derinlemesine öğretilmese de, bu kadar erken adıyla bilinmeye değer dört harf var: Atomicity (bir değişiklik grubu ya hepsi olur ya hiçbiri), Consistency (veritabanı asla kendi kurallarını ihlal eden bir durumda sonlanmaz, var olmayan bir satırı işaret eden bir foreign key gibi), Isolation (bir transaction, başka bir transaction'ın bitmemiş, commit edilmemiş işini görmez), ve Durability (bir değişiklik commit edildiğinde, bir çökmeden sağ çıkar). PostgreSQL dördünü de sağlar -- bu, dersin başındaki "neden özellikle PostgreSQL"in somut cevaplarından biri -- ama tam olarak nasıl, ve bunun eş zamanlı erişim, gerçek kilitleme, ve Spring'in kendi `@Transactional`'ı ("Transaction Management"te zaten işlenen) için pratikte ne anlama geldiği, gerçek SQL ve gerçek eş zamanlı senaryolar onu göstermek için var olana kadar, bilinçli olarak sonraya bırakılıyor.

Best Practices

- Bir Spring Data JPA işlemi opak hissettirdiğinde beş katmanlı yığını aklında tut -- her katman, PostgreSQL'e kadar, belirli, adlandırılabilir bir sonrakine devreder.

- Bir tablonun sütunlarını, gevşek bir alan torbası değil, her satırın uyması gereken bir şekil olarak düşün -- ilişkisel bir veritabanını keyfi yuvalanmış veri saklamaktan temelde farklı kılan şey budur.
- Veriyi "kolaylık için" tablolar arasında çoğaltma dürtüsüne direnç göster -- `category_id` 'nin verisini tekrarlamak yerine `category`'yi işaret etmesinin tüm amacı, tek bir doğruluk kaynağıdır.
- Bir Spring Data JPA çağrısında bir şeyler ters gittiğinde, PostgreSQL'in kendisinin altında gerçek, incelenebilir bir sistem olduğunu hatırla -- bu kursun sonraki dersleri, gerçekte ne yaptığına tam olarak nasıl bakılacağını öğretiyor.

Yaygın Hatalar

- "Veritabanı"nı Hibernate'in tamamen gizlediği bir implementasyon detayı olarak ele almak -- gizlemez; bu kursun tamamı, Spring-ağırlıklı bir kod tabanında bile PostgreSQL'in kendisini anlamanın karşılığını verdiği için var.
- "İlişkisel"in, paylaşılan sütun değerleri üzerinden bağlanan ayrı tablolar fikri yerine, gevşek, günlük anlamda "ilişkili veri" demek olduğunu varsaymak.
- SQL'i (dil) PostgreSQL'le (SQL'i çalıştıran belirli bir program) karıştırmak -- JPA ve Hibernate için zaten işlenen aynı spesifikasyon-vs-implementasyon ayrımı.
- Bu kadar erken ACID'in, transaction'ların, ya da kilitlemenin tam bir açıklamasını beklemek -- bu ders bilinçli olarak yalnızca onları adlandırıyor; "Transactions and Concurrency in PostgreSQL", gerçekten öğretildikleri yer.

Özet, Cheat Sheet ve Terimler Sözlüğü

Özet

- İlişkisel bir veritabanı, veriyi çoğaltmak ya da keyfi olarak yuvalamak yerine, paylaşılan sütun değerleri üzerinden birbirine başvuran tablolar olarak saklar.
- PostgreSQL, ilişkisel bir veritabanının belirli, açık kaynak bir implementasyonudur -- JPA ve Hibernate için zaten işlenen aynı spesifikasyon-vs-implementasyon ilişkisi, bir katman aşağıda.
- Bir tablo, aynı şekildeki satırların adlandırılmış bir koleksiyonudur; bir satır tek bir kayıttır; bir sütun, her satırın bir değere sahip olduğu, adlandırılmış, türlü bir yuvadır.
- Bir Java metot çağrısından saklanmış bir satıra tam yığın: Spring Boot → Spring Data JPA → Hibernate → SQL → PostgreSQL → tablolar/index'ler/kısıtlar/transaction'lar.
- ACID (Atomicity, Consistency, Isolation, Durability) burada yalnızca yüzeysel olarak adlandırılıyor -- "Transactions and Concurrency in PostgreSQL"da tam olarak öğretiliyor.

Cheat Sheet

```
Spring Boot
     ↓
Spring Data JPA        (repository.findBySlug("records"))
     ↓
Hibernate               (SQL'i üretir)
     ↓
   SQL                  (SELECT * FROM topic WHERE slug = 'records')
     ↓
PostgreSQL              (onu çalıştırır, satırı bulur)
     ↓
Tablolar / Index'ler / Kısıtlar / Transaction'lar
```

Terimler Sözlüğü

- **İlişkisel veritabanı (relational database)**: veriyi paylaşılan sütun değerleri üzerinden birbirine başvuran tablolar olarak saklayan bir veritabanı.
- **PostgreSQL**: SQL etrafında inşa edilmiş, ilişkisel bir veritabanının belirli, açık kaynak bir implementasyonu.
- **Tablo / satır / sütun**: aynı şekildeki kayıtların adlandırılmış bir koleksiyonu / tek bir kayıt / her satırın bir değere sahip olduğu, adlandırılmış, türlü bir yuva.
- **ACID**: Atomicity, Consistency, Isolation, Durability -- ilişkisel bir veritabanının transaction'lar için sağladığı dört garanti, bu kursta ileride tam olarak işleniyor.