Transactions and Concurrency in PostgreSQL

psql'de gerçek BEGIN/COMMIT/ROLLBACK, MVCC'nin mekanik açıklaması (xmin/xmax), SELECT ... FOR UPDATE ile satır-seviyesi kilitleme, ve bilerek üretilip açıklanan bir deadlock -- "Transaction Management"in isolation kapsamını TEKRARLAMADAN. Advanced PostgreSQL kategorisinin 4.'ü ve SONUNCUSU -- PostgreSQL kursunun 14/14 topic'ini tamamlıyor.

İleri 30 dk

Son güncelleme: 25 Ağustos 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/transactions-and-concurrency-in-postgresql

---

Spring Data JPA kursundaki "Transaction Management" dersi, `@Transactional`i, üç klasik isolation problemini (dirty/non-repeatable/phantom read), ve PostgreSQL'in kendi isolation varsayılanlarını "Isolation Levels (Kısa Bakış)" ve "PostgreSQL'de Isolation" bölümlerinde zaten kapsamıştı -- hiçbiri burada tekrarlanmıyor. Bu son ders, tamamen `@Transactional`in altına iniyor: `psql`de gerçek `BEGIN` / `COMMIT` / `ROLLBACK`, PostgreSQL'in MVCC'sinin eşzamanlı okuma ve yazmaların bir arada var olmasını gerçekte nasıl sağladığı, `SELECT ... FOR UPDATE` ile gerçek satır-seviyesi kilitleme, ve bilerek üretilip açıklanan bir deadlock.

psql'de ACID: BEGIN, COMMIT, ROLLBACK

Bu kursta şimdiye kadarki her SQL ifadesi kendi başına çalıştı. Bir transaction, birkaç ifadeyi birlikte başarılı ya da başarısız olacak şekilde gruplar:

```
BEGIN;

UPDATE topic SET estimated_minutes = 999 WHERE slug = 'joins';

SELECT estimated_minutes FROM topic WHERE slug = 'joins';
-- 999, aynı transaction içinde görünür

ROLLBACK;

SELECT estimated_minutes FROM topic WHERE slug = 'joins';
-- gerçek değerine geri döndü -- UPDATE, başka herhangi bir oturum açısından hiç olmamış
gibi
```

`BEGIN`, bir transaction başlatır; ondan sonraki her ifade, `COMMIT` onu kalıcı yapana ya da `ROLLBACK` onu -- hiç çalışmamış gibi -- tamamen atana kadar geçicidir. Bu, doğrudan görünür

kılınmış atomicity'dir -- ACID'deki "A" -- yukarıdaki `UPDATE`, transaction *içinde* gerçek ve okunabilirdi, sonra `ROLLBACK` tarafından tamamen geri alındı, arkasında hiçbir kısmi iz bırakmadan.

Autocommit: Açık Bir Transaction Olmadan Ne Olur

Bu kursta daha önceki her tek-ifadeli örnek -- her `SELECT`, her `INSERT` -- hiç görünür bir `BEGIN` / `COMMIT` olmadan çalıştı, çünkü PostgreSQL, açık bir transaction içinde olmayan herhangi bir ifadeyi kendi örtük transaction'ının içine sarar, başarılı olursa hemen commit eder. "Inserting, Updating, and Deleting Data"nın gerçek migration `INSERT`lerinin hiç açık bir `BEGIN`e ihtiyaç duymamasının nedeni budur -- her biri zaten kendi tek-ifadeli transaction'ı olarak çalıştı. `BEGIN` / `COMMIT` / `ROLLBACK`, yalnızca birden fazla ifadenin gerçekten bir birim olarak başarılı ya da başarısız olması gerektiği anda gerekli hâle gelir -- "Transaction Management"te zaten kapsanan `@Transactional`in, hiçbir uygulama kodu `BEGIN` / `COMMIT`i kendisi hiç yazmadan Java tarafından tam olarak başardığı şey.

MVCC: PostgreSQL Eşzamanlı Okumaları Bloklamadan Nasıl Mümkün Kılar

"PostgreSQL'de Isolation" MVCC'yi `REPEATABLE_READ` / `SERIALIZABLE`in arkasındaki mekanizma olarak zaten adlandırmıştı, gerçekte nasıl çalıştığını açıklamadan -- işte mekanizmanın kendisi. **MVCC** (Multi-Version Concurrency Control), PostgreSQL'in bir satırı güncellendiğinde yerinde asla üzerine yazmadığı anlamına gelir -- bunun yerine satırın *yeni bir versiyonunu* yazar ve eski versiyonu üzeri-çizilmiş olarak işaretler, hiçbir şey eski olana artık ihtiyaç duyamayana kadar ikisini de tutar. Her satır gizlice iki gizli sistem kolonu taşır, `xmin` (bu satır versiyonunu oluşturan transaction'ın id'si) ve `xmax` (varsa, onu silen ya da üzerine çizen transaction'ın id'si) -- bir `SELECT` bunları varsayılan olarak asla görmez, ama PostgreSQL'in belirli bir transaction'ın hangi satır versiyonunu görmesine izin verildiğine karar vermek için gerçekte danıştığı şey bunlardır.

Bu, PostgreSQL'de okuyucuların yazıcıları asla bloklamamasının, ve yazıcıların okuyucuları asla bloklamamasının tam nedenidir -- `topic`i okuyan bir transaction, başka bir transaction eşzamanlı olarak yeni versiyonlar yazıyor olsa bile, kendi snapshot'ına göre var olan hangi satır versiyonları varsa onları görür; bir kuyruk yok, bekleme yok, yalnızca aynı mantıksal satırın iki farklı (ama ikisi de tamamen geçerli) versiyonuna bakan iki transaction var. Eski bir satır versiyonu, yalnızca hiçbir çalışan transaction ona artık ihtiyaç duyamayacak hâle geldiğinde fiziksel olarak temizlenir -- PostgreSQL'in arka plan `VACUUM` sürecinin ele aldığı iş, bu projenin kendi ölçeği için doğrudan çalıştırmaya gerek olmadan var olduğunu bilmeye değer bir ayrıntı.

Satır-Seviyesi Kilitleme: SELECT ... FOR UPDATE

MVCC eşzamanlı okumaları zarifçe ele alır, ama gerçekten aynı satırı *değiştirmeye* çalışan iki transaction hâlâ koordine olmalıdır -- satır-seviyesi kilitlemenin amacı budur. `SELECT ...`

`FOR UPDATE`, döndürdüğü satırları kilitler, başka herhangi bir transaction'ın o aynı satırları, ilk transaction commit ya da rollback edene kadar değiştirmesini (ya da onları da kilitlemesini) engeller:

```
-- Oturum A
BEGIN;
SELECT * FROM topic WHERE slug = 'joins' FOR UPDATE;
-- 'joins' için topic satırı şimdi Oturum A tarafından kilitli

-- Oturum B, eşzamanlı çalıştırılan, Oturum A'nın transaction'ı hâlâ açıkken
UPDATE topic SET estimated_minutes = 20 WHERE slug = 'joins';
-- Oturum B burada bloklanır -- Oturum A commit ya da rollback edene kadar bekler
```

Oturum B'nin `UPDATE`i başarısız olmaz -- yalnızca bekler, gerçekten bloklanmış, Oturum A'nın transaction'ı bir şekilde sona erene kadar. Bu, Spring Data JPA kursundaki "The Persistence Context and Locking"in zaten `@Lock(LockModeType.PESSIMISTIC_WRITE)` olarak kapsadığı şeyin altındaki SQL-seviyesi mekanizmadır -- Hibernate, o annotation'ın altında tam olarak bu `FOR UPDATE` cümlesini verir, daha egzotik hiçbir şey değil.

Gerçek Bir FOR UPDATE Senaryosu

`SELECT ... FOR UPDATE`, özellikle bir okumanın, az önce okunana bağlı bir yazmayla takip edilmesi gerektiğinde ve arada başka hiçbir transaction'ın o değeri değiştirememesi gerektiğinde yerini hak eder -- klasik gerçek bir durum, sınırlı bir miktarı azaltmaktır:

```
BEGIN;
SELECT sort_order FROM topic WHERE slug = 'joins' FOR UPDATE;
-- sort_order'ı oku, onunla bir uygulama-seviyesi hesaplama yap
UPDATE topic SET sort_order = sort_order + 1 WHERE slug = 'joins';
COMMIT;
```

`FOR UPDATE` olmadan, iki eşzamanlı transaction ikisi de aynı `sort_order`ı okuyabilir, ikisi de aynı "sonraki" değeri bağımsız olarak hesaplayabilir, ve ikisi de onu yazabilir -- gerçek bir kaybolan güncelleme, çünkü ikinci yazma, hiçbir transaction diğerinin olduğunu bilmeden ilkinin üzerine sessizce yazar. `FOR UPDATE`, tam olarak bunu, ikinci transaction'ın kendi `SELECT ... FOR UPDATE`sini, ikisinin de eski bilgiyle devam etmesine izin vermek yerine ilkinin bitmesini beklemeye zorlayarak önler.

Bir Deadlock Üretmek ve Açıklamak

Bir **deadlock**, iki transaction'ın her biri diğerinin ihtiyaç duyduğu bir kilidi tuttuğunda, ve her biri diğerinin onu serbest bırakmasını beklediğinde olur -- hiçbiri asla ilerleyemez:

```
-- Oturum A
BEGIN;
UPDATE topic SET estimated_minutes = 1 WHERE slug = 'joins';
-- Oturum A şimdi 'joins' satırı üzerinde bir kilit tutuyor

-- Oturum B, eşzamanlı
BEGIN;
UPDATE topic SET estimated_minutes = 1 WHERE slug = 'aggregation-and-group-by';
-- Oturum B şimdi 'aggregation-and-group-by' satırı üzerinde bir kilit tutuyor

-- Oturum A, sonraki
UPDATE topic SET estimated_minutes = 2 WHERE slug = 'aggregation-and-group-by';
-- Oturum A bloklanır, Oturum B'nin kilidini bekler

-- Oturum B, sonraki
UPDATE topic SET estimated_minutes = 2 WHERE slug = 'joins';
-- Oturum B de bloklanırdı, Oturum A'nın kilidini bekleyerek --
-- PostgreSQL döngüyü önce tespit etmeseydi
```

PostgreSQL, iki oturumun sonsuza kadar beklemesine izin vermek yerine bu döngüyü aktif olarak tespit eder -- iki transaction'dan biri ("kurban" olarak seçilen, tipik olarak geri almanın daha ucuz olacağı) gerçek bir hata alır (`deadlock detected`) ve otomatik olarak geri alınır, kilitlerini serbest bırakarak diğer transaction'ın ilerlemesine izin verir. Düzeltme bir veritabanı ayarı değil -- bir kodlama disiplinidir: birden fazla satır üzerinde kilit almayı, onlara dokunan her transaction genelinde her zaman aynı, tutarlı sırada yap (burada, her zaman `aggregation-and-group-by` den önce `joins` , asla tersi değil), ki bu, bu örneğin ürettiği döngüsel beklemeyi yapısal olarak imkânsız kılar.

Yaygın Yanlış Anlamalar

"Bir transaction tüm tabloyu kilitler." Varsayılan olarak değil -- `SELECT ... FOR UPDATE` , yalnızca döndürdüğü belirli satırları kilitler; MVCC, sıradan okumaların hiç kimse için hiçbir şeyi asla kilitlemediği anlamına gelir. **"MVCC, PostgreSQL'in kilitlere ihtiyacı olmadığı anlamına gelir."** Kilitlere ne sıklıkta ihtiyaç duyulduğunu azaltır (okuyucular ve yazıcılar asla birbirini bloklamaz), ama aynı satır üzerindeki gerçek yazma-yazma çatışmaları hâlâ satır-seviyesi kilitlemeye ihtiyaç duyar, tam olarak bu dersin `FOR UPDATE` örneklerinin gösterdiği gibi. **"Bir deadlock, veritabanının bozuk olduğu anlamına gelir."** Tam tersi -- deadlock tespiti, PostgreSQL'in çözülemez bir döngüyü doğru şekilde fark edip otomatik olarak çözmesidir, alternatifi (iki transaction'ın da sonsuza kadar donması) gerçekten bozuk olan sonuç olurdu.

Best Practices

- Özellikle bir okumanın sonucu doğrudan bir yazmayı bilgilendirdiğinde, ve arada başka hiçbir transaction'ın o değeri değiştirememesi gerektiğinde `SELECT ... FOR UPDATE` e

başvur -- bu dersteki "oku, sonra artır" deseni kanonik durumdur, ve "The Persistence Context and Locking"te zaten kapsanan `@Lock(PESSIMISTIC_WRITE)` in altında güvendiği aynı mekanizmadır.
- Birlikte dokunabilecekleri her transaction genelinde birden fazla satırı tutarlı bir sırada kilitle -- bu dersin deadlock örneği, özellikle iki oturum `joins` ve `aggregation-and-group-by` yi ters sırada kilitlediği için var; her zaman aynı sırayı seçmek döngüyü tamamen ortadan kaldırır.
- Transaction'ları, özellikle satır kilidi tutanları, kısa tut -- `BEGIN` ile `COMMIT` / `ROLLBACK` arasındaki her ifade, başka bir transaction'ın bunun tuttuğu bir kilidi beklemek için harcayabileceği zamandır.
- Bu projenin kendi repository'lerinin hiçbir yerde `SELECT ... FOR UPDATE` ya da açık satır kilitleme kullanmadığını tanı -- salt-okunur ağırlıklı doğasının gerçekten dürüst bir yansıması ("Transaction Management"in bu kod tabanının bazı bölümleri hakkındaki kendi "hâlâ neden `@Transactional` yok" gözlemini yansıtan), bir gözden kaçırma değil.

Yaygın Hatalar

- Bir `ROLLBACK` in, herhangi bir şey geri alınmadan önce isimle istenmesi gerektiğini varsaymak -- bir istemcinin basitçe bağlantısını kestiği commit edilmemiş bir transaction da otomatik olarak geri alınır, ama açık bir `ROLLBACK` yerine buna güvenmek, niyeti daha sonra okumayı çok daha zor kılar.
- Bir `SELECT ... FOR UPDATE` kilidini, veritabanıyla ilgisi olmayan yavaş bir işlem boyunca (harici bir API çağrısı, uzun bir hesaplama) açık tutmak -- o satıra ihtiyaç duyan her diğer transaction, yalnızca veritabanı işi için değil, tüm işlem için bekler.
- Bir deadlock'un pratikte göz ardı edilecek kadar nadir olduğunu varsaymak -- iki işlemin gerçekten aynı satırları farklı sıralarda kilitleyebildiği herhangi bir kod yolu, ne kadar nadir olursa olsun, gerçek eşzamanlı yük altında sonunda deadlock olur; birden fazla satıra dokunan uygulama kodu, bunun olamayacağı varsayımı değil, tam olarak bu hata için bir yeniden deneme stratejisine ihtiyaç duyar.
- Bir `SELECT ... FOR UPDATE` bloğunu bir sorguyu bloklayan bir `Seq Scan` ile karıştırmak -- "Indexes and Query Performance with EXPLAIN", tarama türlerini bir maliyet/performans kavramı olarak kapsadı; burada kapsanan kilitleme, ondan tamamen bağımsız bir doğruluk/eşzamanlılık kavramı.

Özet, Cheat Sheet ve Terimler Sözlüğü

Özet

- `BEGIN` / `COMMIT` / `ROLLBACK`, atomicity'yi `psql` de doğrudan görünür kılar; açık bir transaction dışındaki her ifade, zaten PostgreSQL'in kendi örtük, otomatik-commit'lenen transaction'ı içinde çalışır.
- MVCC, bir güncellemenin yerinde üzerine yazmak yerine yeni bir satır versiyonu yazması anlamına gelir, gizli `xmin` / `xmax` kolonları üzerinden takip edilir -- PostgreSQL'de sıradan

okumaların yazmaları, ya da tam tersini, asla bloklamamasının nedeni budur, "PostgreSQL'de Isolation"ın açıklamadan adlandırdığı mekanizma.

- `SELECT ... FOR UPDATE`, belirli satırları bir transaction süresince kilitler, başka transaction'ların o sona erene kadar onları değiştirmesini engeller -- "The Persistence Context and Locking"te zaten kapsanan `@Lock(PESSIMISTIC_WRITE)` in altındaki tam SQL.
- Bir deadlock, iki transaction'ın her biri diğerinin ihtiyaç duyduğu bir kilidi tuttuğunda olur; PostgreSQL döngüyü tespit eder ve bir transaction'ı otomatik olarak geri alır -- birden fazla satırı tutarlı bir sırada kilitleyerek tamamen önlenir.
- Bu projenin kendi repository'leri hiçbir yerde açık satır kilitleme kullanmaz, salt-okunur ağırlıklı iş yükünün bir boşluk değil gerçek bir yansıması -- "Transaction Management"in `@Transactional` in kendisine zaten uyguladığı aynı dürüst desen.

Cheat Sheet

```
BEGIN;
...ifadeler...
COMMIT;    -- ya da ROLLBACK;

SELECT ... FROM t WHERE ... FOR UPDATE;  -- commit/rollback'e kadar eşleşen satırları
kilitler
```

Terimler Sözlüğü

- **Atomicity**: bir transaction'ın ifadelerinin tek bir birim olarak birlikte başarılı ya da başarısız olması -- ACID'deki "A".
- **MVCC (Multi-Version Concurrency Control)**: PostgreSQL'in yerinde üzerine yazmak yerine birden fazla satır versiyonu tutma stratejisi, okumaların ve yazmaların birbirini bloklamadan ilerlemesine izin verir.
- **xmin / xmax**: hangi transaction'ın belirli bir satır versiyonunu oluşturduğunu ve (varsa) üzerine çizdiğini kaydeden gizli sistem kolonları.
- **Satır-seviyesi kilit**: belirli satırlar üzerinde tutulan (`SELECT ... FOR UPDATE` ile), serbest bırakılana kadar başka transaction'ların onları değiştirmesini engelleyen bir kilit.
- **Deadlock**: her biri diğerinin tuttuğu bir kilidi bekleyen transaction'ların bir döngüsü, PostgreSQL'in birini otomatik olarak geri alarak tespit edip çözdüğü.