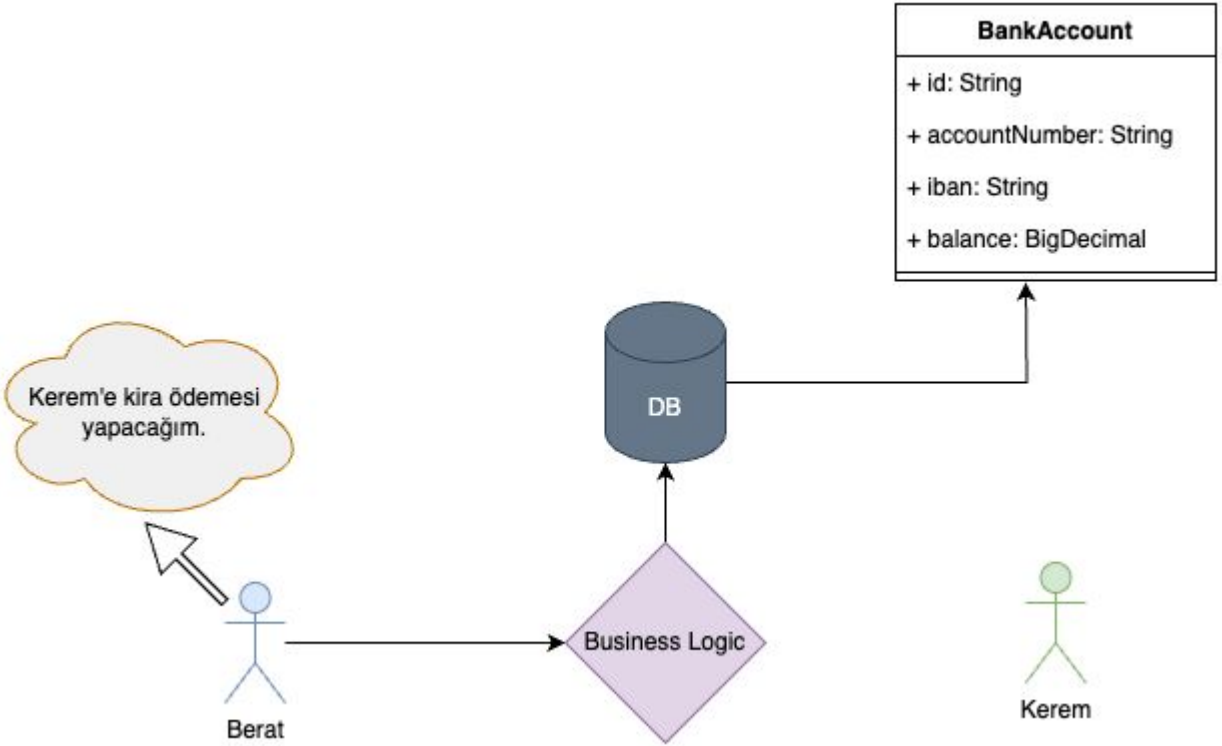
# Transaction Management

Berat Yesbek

# Transaction Management Nedir?

Bir işlemin gerçekleşmesi sırasında meydana gelebilecek hataları ve süreçleri yönetmedir


BankAccount
+ id: String
+ accountNumber: String
+ iban: String
+ balance: BigDecimal
DB
Kerem'e kira ödemesi yapacağım.
Business Logic
Berat
Kerem

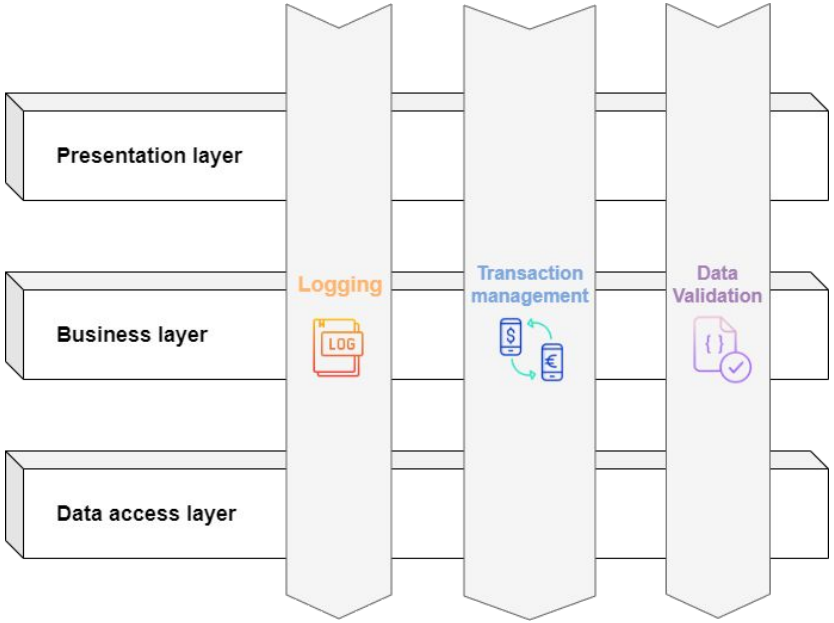
# @Transactional

Bu annotasyon Spring Framework’ de veri tabanı işlemlerini kapsayan ve onlar için bir transaction management yapan annotasyondur. Sınıflara ve Metot’lara tanımlanır.

Spring Boot’ da transaction management enabled olarak gelir. Spring tarafında ise konfigürasyon gereklidir.

# Derin Bakış - 1

Transaction yapısı AOP (Aspect Oriented Programing) kullanarak uygulanır . AOP cross-cutting-concern' leri (endişeleri) iş mantığından ayırır.



# Derin Bakış - 2

Spring dynamic proxy kullanır. @Transactional olarak işaretlenmiş nesneler için proxy ler oluşturur. Bir fonksiyonun öncesinde veya sonrasında araya girerek transaction için gerekli olan business logic işler.

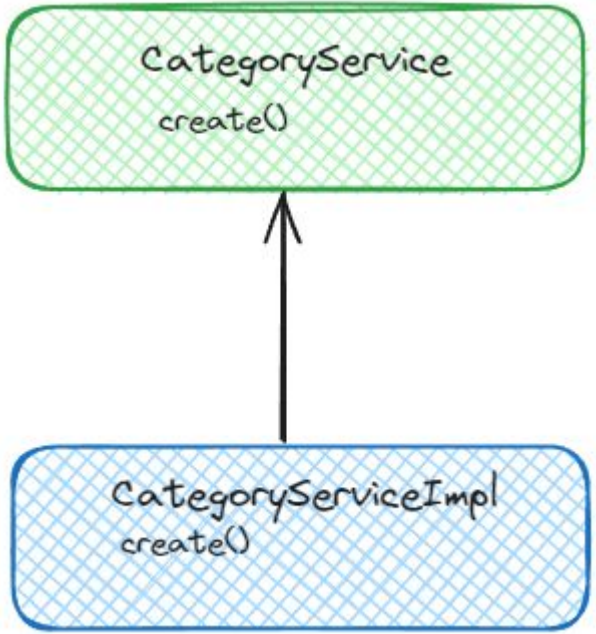
# Proxy Nedir?

Bir proxy, orijinal nesneye erişimi kontrol eder ve istek orijinal nesneye ulaşmadan önce veya sonra bir şey gerçekleştirmenize izin verir.

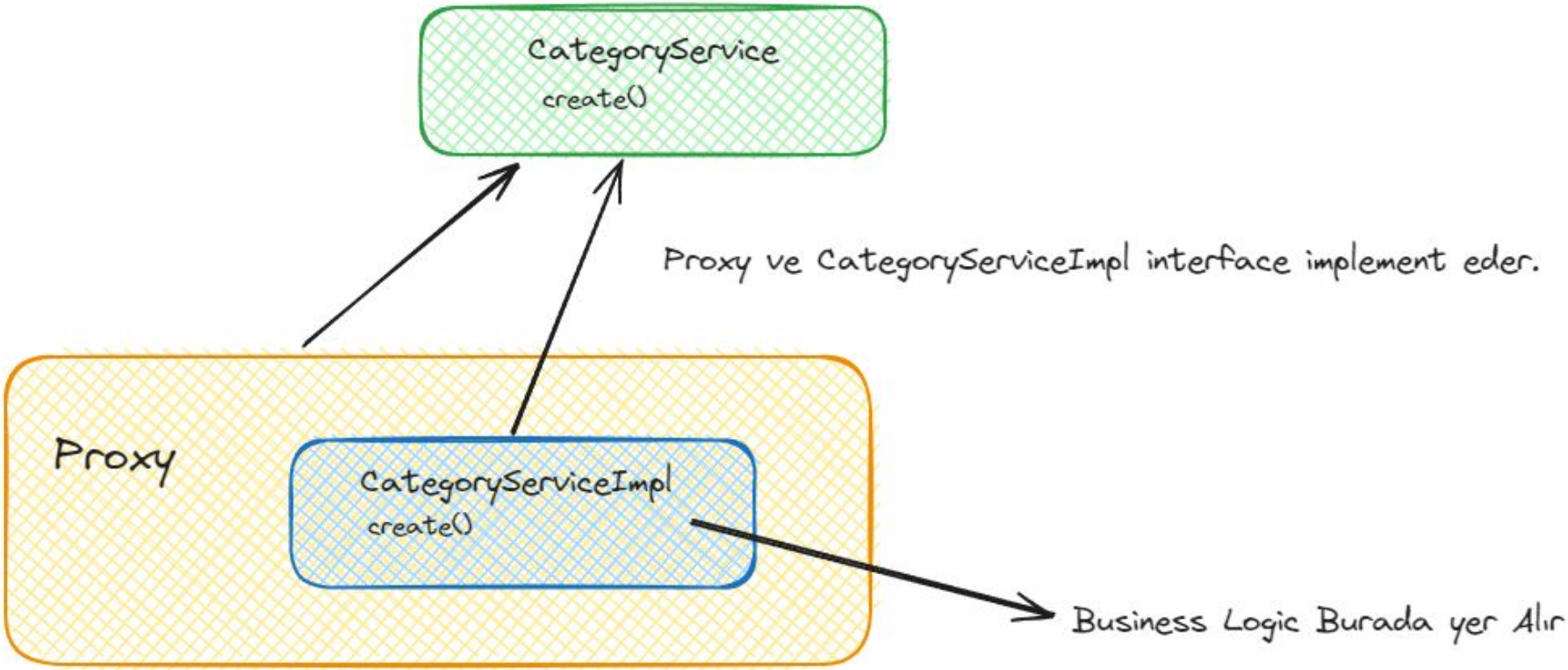
Aynı zamanda orjinal nesne için yedekleme yapar

# Transaction Proxy Örnek - 1

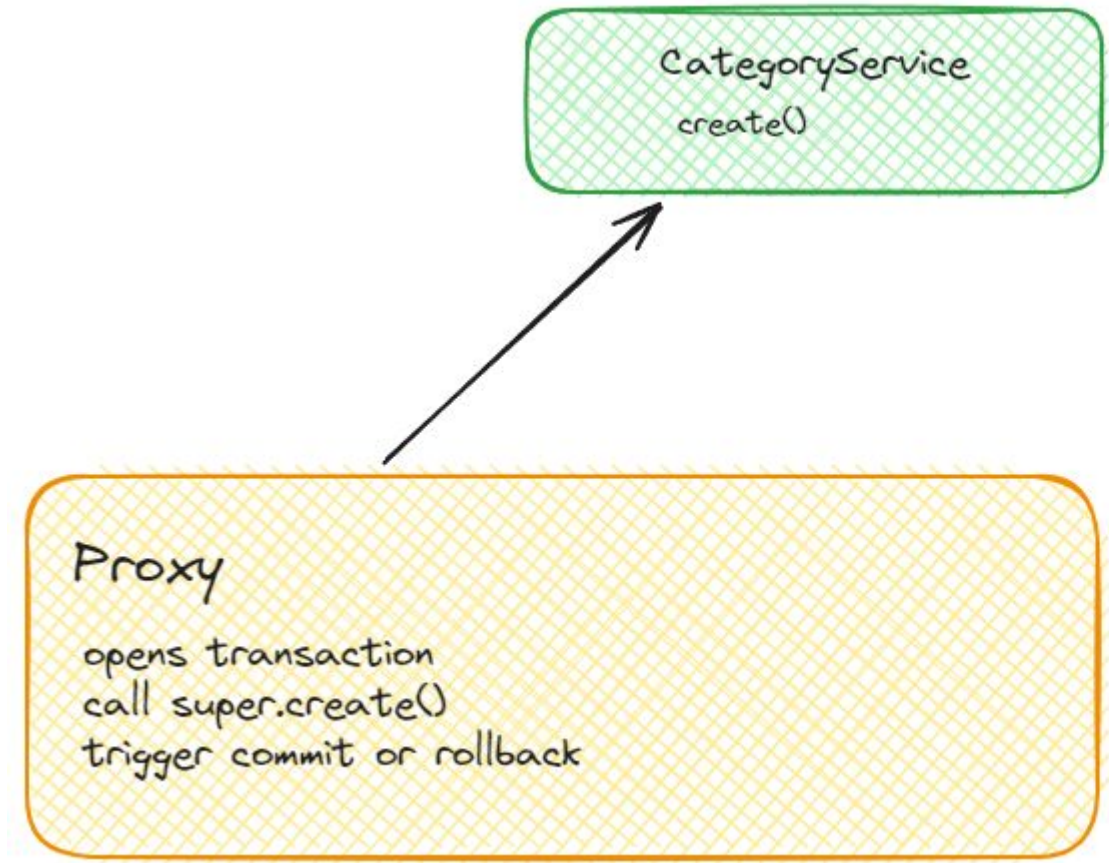
Bu sizce geçerli bir örnek midir?



# Transaction Proxy Örnek - 2



# İmplementation Yoksa ?




Çalışma zamanında oluşur

# Propagation

Propagation Spring Transaction’ın nasıl davranacağını belirleyen enum yapısıdır.

- REQUIRED
- SUPPORTS
- MANDATORY
- REQUIRES_NEW
- NOT_SUPPORTED
- NEVER
- NESTED

# Propagation Type - REQUIRED

Default Type olarak gelir. 20 adet veri kaydetmek istediğinizi varsayalım 19. da hata attı tüm işlemler geri alınır. En çok kullanılan türdür. Mevcut bir Transaction varsa kendisine dahil eder.

# Propagation Type - REQUIRES_NEW

Daha öncesinde bir session varsa askıya alır ve yeni bir session açar. Burada ki Transactionın diğerlerinin etkilememesini sağlamaktır.

# Propagation Type - SUPPORTS

Eğer kendisinden önce bir Transaction başlatılmış ise ona dahil olur yoksa non-transactional davranarak yoluna devam eder.

# Propagation Type - NOT_SUPPORTED

Çağrıldığı noktadan itibaren tüm yapılar non-transactional olarak çağrılır.
Kendisinden önce bir Transaction varsa askıya alır.

# Propagation Type - NEVER

Kendinden önce bir Transaction başlatılırsa hata fırlatır.

- Bağımsız fonksiyonlarda kullanılmalıdır

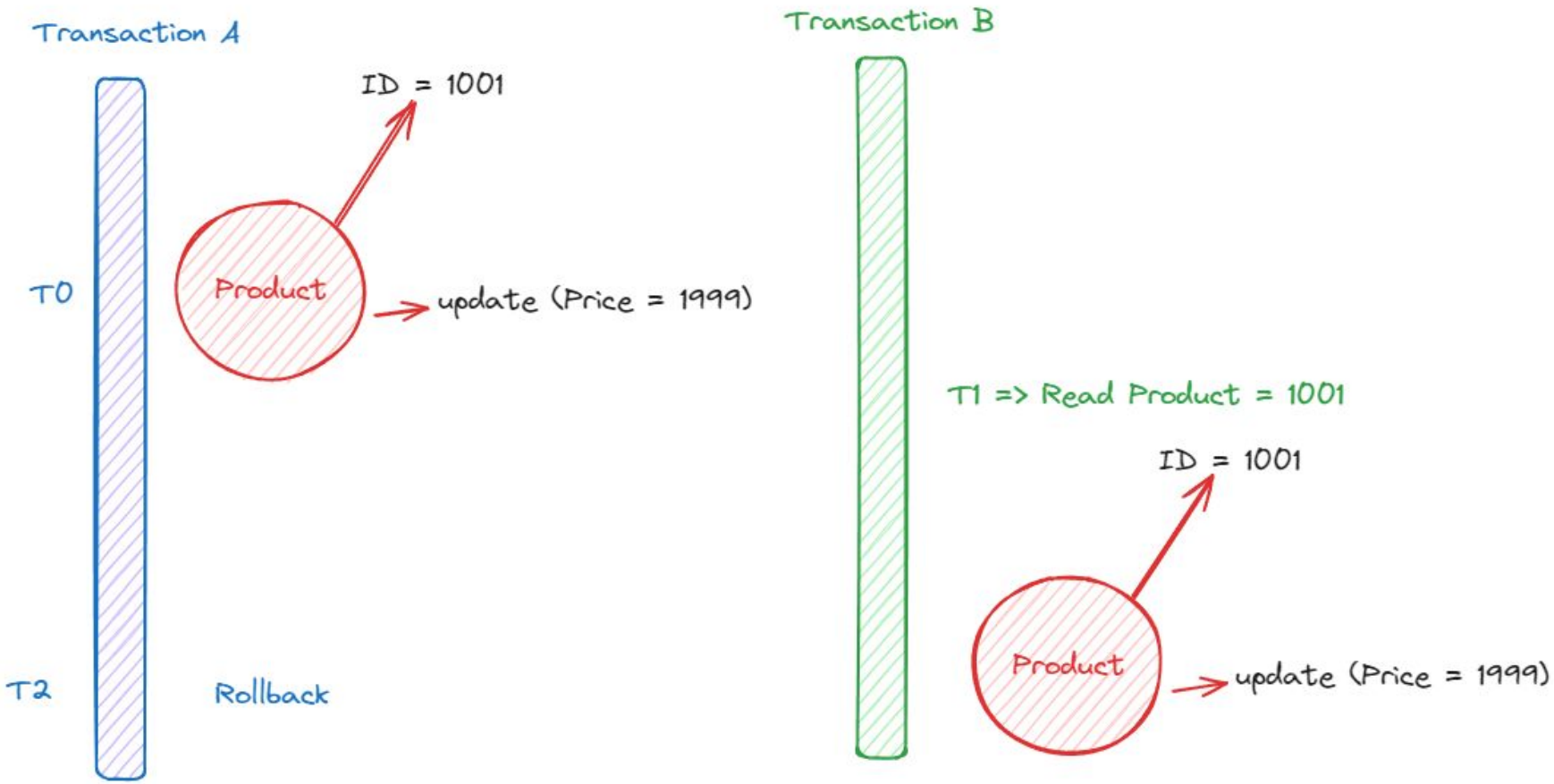
# Propagation Type - MANDATORY

Kendinden önce mutlaka bir Transaction dahil olması istenir. Aksi takdirde hata fırlatır. Her zaman REQUIRED gibi davranır.

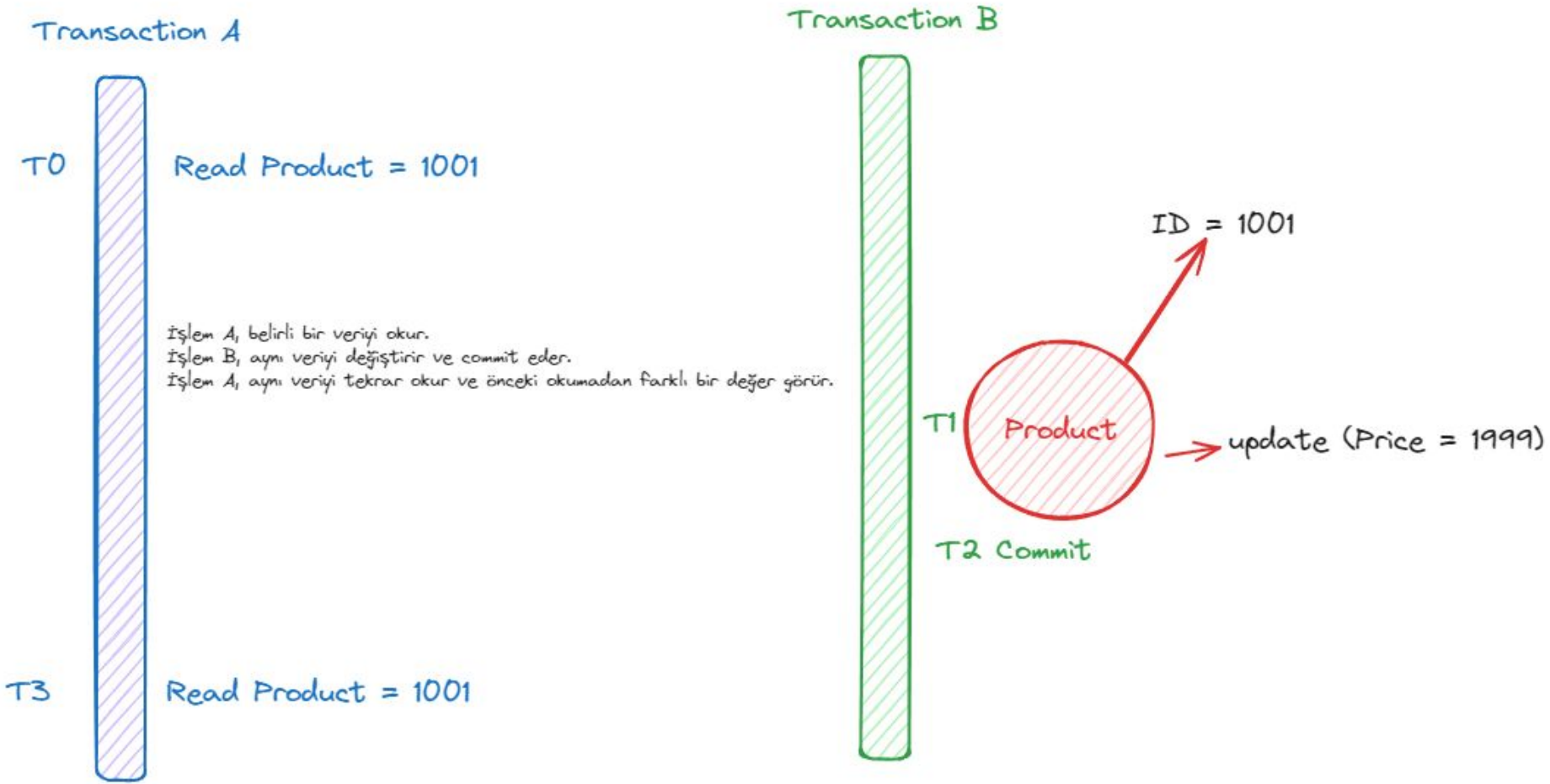
# ISOLATION LEVEL

Eş zamanlı işlemlerde verinin tutarlılığını sağlayan. Ve izole eden bir yapıdır. Verinin bütünlüğünü korur.

# Dirty Read


Transaction A
Transaction B
ID = 1001
T0
Product
update (Price = 1999)
T1 => Read Product = 1001
ID = 1001
Product
update (Price = 1999)
T2
Rollback

# NON REPEATABLE READS



# PHANTOM READS

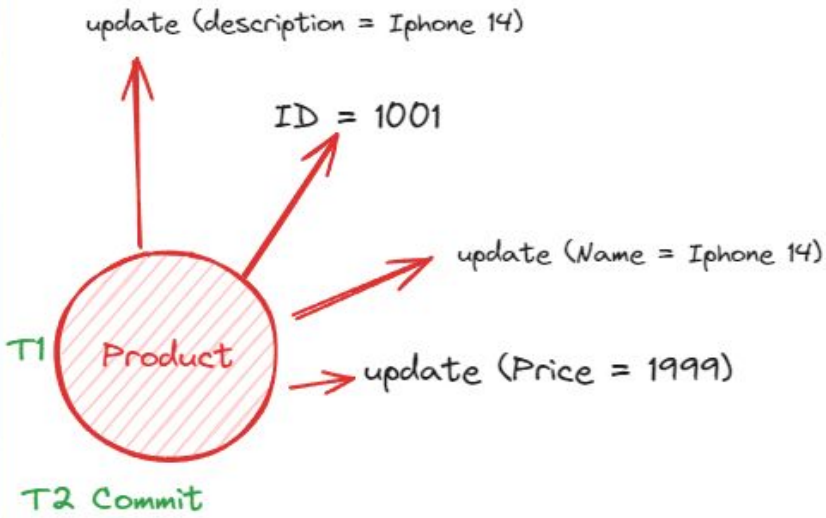
Transaction A

T0 Read Product = 1001

İşlem A, belirli bir sorgu çalıştırarak bir veri kümesini okur.
İşlem B, aynı sorguya göre bir veya birkaç yeni veri ekler ve commit eder.
İşlem A, aynı sorguyu bir kez daha çalıştırır ve farklı bir dateset elde eder

T3 Read Product = 1001

Transaction B



# READ_UNCOMMITTED

en düşük izolasyon seviyesidir ve bir çok eşzamanlı erişime izin verir.

Eşzamanlılık ile ilgili problemleri önleyemez.

# READ_COMMITTED

2. Seviye izolasyon dur. DIRTY READ engeller. Ama eşzamanlılığın diğer problemleri gerçekleşebilir.

# REPEATABLE_READ

3. seviye izolasyon dur. DIRTY READ ve NON REPEATABLE READ engeller.

# SERIALIZABLE

En üst seviye izalasyondur. Tüm eşzamanlılık problemlerini engeller. Eşzamanlı çağrıları sırasıyla yürütür. Performans açısından pahalıdır.