**Pusula Talent Academy 2025 - SQL & DBA Case Study**

# Question 1: Performance & Scalability Analysis in Hospital Data

Scenario:

A table below has been used for 5 years in a Hospital Information Management System (HBYS). Each day, about 25,000 rows are inserted.

Recently, queries on this table have become slower and users have reported difficulty accessing past records.

CREATE TABLE HastaIslemLog (

Id INT IDENTITY(1,1) PRIMARY KEY,

HastaId INT,

IslemTarihi DATETIME,

IslemKodu NVARCHAR(20),

Aciklama NVARCHAR(500)

);

Question:

1. What could be the reasons for the performance degradation?

* **Index Eksikliği:** Tabloya sadece **Id** kolonu üzerinden ***clustered index*** tanımlı. Halbuki sorgularda genelde **HastaId**, **IslemTarihi** veya **IslemKodu** gibi alanlarda filtreleme yapılıyor. Bu kolonlarda ***non clustered index*** olmayınca ***index seek*** yerine ***table scan*** yapılıyor. Bu da maliyeti artırarak performansı ciddi şekilde düşürüyor.
* **Tablo Boyutunun Şişkinliği:** 5 yıl boyunca her gün 25.000 satır eklendiğinde toplam kayıt sayısı 45 milyona çıkıyor. Böyle büyük tablolar üzerinde index ve arşivleme yapılmazsa sorguların süresi uzar, bakım ve yedekleme işlemleri ağırlaşır.
* **Geniş Sütun Kullanımı: Aciklama NVARCHAR(500)** sütunu çok geniş tanımlandığı için her satır fazla yer kaplıyor. Bu durum 8 KB’lık aynı sayfaya daha az satır sığmasına neden oluyor ve bellekte daha çok yer tüketiyor. Çoğu sorguda bu sütun kullanılmasa da tablo her okunduğunda beraberinde geliyor.
* **İstatistiklerin Güncel Olmaması:** Bu kadar fazla INSERT işlemi yapılan bir tabloda ***statistics*** hızlıca eskiyebilir. İstatistikler güncel değilse SQL Server yanlış ***execution plan*** seçebilir ve sorgular yavaş çalışabilir.
* **Index Fragmentation:** Çok fazla INSERT işlemi eğer varsa ***non clustered index***’lerde parçalanmaya yol açabilir. Bu da daha fazla sayfa okunmasına ve performans kaybına neden olur.
* **Ad Hoc Query Kullanımı:** Eğer sorgular parametreli değil de her defasında farklı değerlerle gönderiliyorsa ***execution plan cache*** verimli kullanılmaz. Bu da ***plan cache pollution*** sorununa sebep olur ve performansı olumsuz etkiler.

1. What improvements would you suggest for better sustainability?

* **Non Clustered Index’lerin Doğru Şekilde Tanımlanması: HastaId**, **IslemTarihi** ve **IslemKodu** gibi sık sorgulanan sütunlar üzerinde ***composite non clustered index*** oluşturulmalı. Böylece ***table scan*** yerine ***index seek*** yapılması sağlanır ve sorgular daha hızlı döner. Ayrıca ***Key Lookup*** azaltmak için sorgularda sık kullanılan sütunları index’e INCLUDE etmeliyiz.
* CREATE NONCLUSTERED INDEX IX\_HastaIslemLog\_HastaId\_IslemTarihi

ON HastaIslemLog (HastaId, IslemTarihi DESC)

INCLUDE (IslemKodu)

* **Aciklama Sütun Yapılandırılması:** NVARCHAR(500) kullanımı tabloyu çok genişletiyor bu yüzden mümkünse uzunluk daha küçük tutulmalı ya da bu alan ayrı bir detay tablosuna taşınmalı. Böylece sadece gerektiğinde ***JOIN*** ile çağrılır.
* **Tablonun Bölünmesi veya Arşivlenmesi:** 5 yıl boyunca tek tabloda veri tutmak performansı olumsuz etkiliyor. Eski kayıtlar ayrı arşiv tablolarına taşınabilir veya **IslemTarihi** üzerinden ***partitioning*** yapılabilir. Bu sayede hem sorgular daha hızlı çalışır hem de bakım işleri kolaylaşır.
* **İstatistiklerin Düzenli Güncellenmesi:** Yoğun INSERT sebebiyle istatistikler çabuk eskimiş olabilir. Bu yüzden günlük ya da haftalık ***job*** ile ***UPDATE STATISTICS*** çalıştırmak faydalı olur.
* **Index Bakımının Yapılması:** Zamanla index’lerde fragmentation oluşur. %10-30 arası parçalanmada ***Reorganize*** %30 üzeri parçalanmada ***Rebuild*** işlemleri yapılmalı. Bu işlemler sayesinde index performansı korunur.
* **Parametreli Sorgular Kullanılması:** Ad Hoc Query kullanımından kaçınılması. Parametreli sorgular ***execution plan***’ı tekrar kullanır bu da ***plan cache pollution*** sorununu azaltır.

1. Do you think using the table in this way for 5 years was the correct approach? Why or why not?

* Hayır doğru bir yaklaşım değil. Bu tablo ilk kurulumda pratik bir çözüm olmuş olabilir ama 5 yıl boyunca hiç arşivleme, partitioning veya doğru index’leme yapılmadan kullanılması uzun vadede doğru bir tasarım değil. Eğer baştan itibaren bakım ve veri yönetimi planı olsaydı, sistem hem performanslı hem de daha kolay yönetilebilir olurdu.
* **Neden Doğru Değil ?**   
  Tabloda sadece **Id** üzerinde **clustered index** bulunması başlangıçta INSERT işlemleri için uygun olsa da sorgular genelde HastaId ve IslemTarihi kolonları üzerinden çalıştığı için bu alanlarda doğru index’ler olmadığından ***table scan*** yapmak zorunda kalıyor ve bu da sorguları yavaşlatıyor. Ayrıca yıllar boyunca hiçbir arşivleme yapılmadığı için hem güncel hem de eski veriler aynı tabloda tutuluyor bu durum sorguların daha fazla satır üzerinde çalışmasına ve bakım sürelerinin uzamasına sebep oluyor. Büyük ve sürekli değişen bu tabloda istatistiklerin güncel tutulmaması yanlış ***execution plan*** seçilmesine sebep olarak performansı olumsuz etkiliyor. Tablo büyüdükçe fragmentation artıyor bu da index bakımını zorlaştırıyor.

Note: Open-ended. Focus on reasoning, not just query writing.

# Question 2: Index Strategy & Query Optimization Thinking

Scenario:

The following query is frequently used by end users:

SELECT \*

FROM HastaKayit

WHERE LOWER(AdSoyad) LIKE '%ahmet%' AND YEAR(KayitTarihi) = 2024

Question:

1. What performance problems might arise from this query?

* Verilen sorguda bazı noktalar performans sorunlarına yol açabilir. Öncelikle SELECT \*ifadesi gereksiz sütunları da getirdiği için satır boyutunu artırır ve I/O maliyetini yükseltir. LOWER(AdSoyad) ve YEAR(KayitTarihi) gibi fonksiyonların sütunlar üzerinde kullanılması sorguyu **Sargable** olmaktan çıkararak index’lerden faydalanamaz hale getirir. Bu durumda mevcut index’ler kullanılamaz ve ***table scan*** yapmak durumunda kalınır. Bu sorgu parametrik yazılmadığı için her defasında yeni bir ***execution plan*** oluşturulur ve ***plan cache*** gereksiz yere şişerek belleği doldurur.

1. How would you optimize this query and/or the table structure?

* SELECT \* yerine sadece ihtiyaç duyulan sütunları seçmek gereksiz veri taşınmasını engeller ve I/O maliyetini azaltır. **YEAR(KayitTarihi)** fonksiyonu yerine   
  WHERE KayitTarihi >= @StartDate AND KayitTarihi < @EndDate   
  gibi tarih aralığı kullanmak daha doğru olur çünkü bu sayede index’lerden faydalanılabilir. **LOWER(AdSoyad)** fonksiyonu yerine tabloya   
  ALTER TABLE HastaKayit ALTER COLUMN AdSoyad NVARCHAR(100) COLLATE Turkish\_CI\_AI  
  komutuyla collate eklenerek ***index seek*** yapılabilir ve performans artar. Ayrıca sorgunun parametrik hale getirilmesi çok önemlidir çünkü parametrik sorgular ***execution plan***’ın tekrar kullanılmasını sağlar ve ***plan cache*** daha verimli olur. Son olarak **KayitTarihi** ve **AdSoyad** sütunları üzerinde doğru index’lerin tanımlanması da bu sorgunun performansını büyük ölçüde iyileştirir.

1. Are there any improvements that could be made on the application side?

* Uygulama tarafında yapılabilecek bazı iyileştirmeler de sorgu performansını doğrudan etkiler. Öncelikle sorguların parametreli gönderilmesi gerekir böylece her defasında yeni ***execution plan*** üretilmez ve ***plan cache*** daha verimli kullanılır. SELECT \* yerine sadece ihtiyaç duyulan sütunların seçilmesi de hem ağ trafiğini azaltır hem de I/O maliyetini düşürür. **YEAR()** fonksiyonu yerine başlangıç ve bitiş tarihleri parametre olarak gönderilmeli böylece sorgu uygun index’lerden faydalanabilir. Büyük veri setleri tek seferde çekilmemeli bunun yerine ***paging*** kullanılmalıdır. Ayrıca connection pooling kullanılması ve timeout değerlerinin ayarlanması SQL Server kaynaklarının verimli kullanılmasını sağlar. Sık yapılan aynı sorgular için uygulama tarafında **cache** kullanmak da veritabanı yükünü azaltır.

Note: Expect analysis and optimization suggestions.

# Question 3: T-SQL Query Challenge (Hospital Sales Example)

Scenario:

In the HBYS system, the hospital pharmacy sells products. Sales and product details are stored in the following tables:

CREATE TABLE Urun (

UrunID INT PRIMARY KEY,

UrunAdi NVARCHAR(100),

Fiyat DECIMAL(10,2)

);

CREATE TABLE Satis (

SatisID INT PRIMARY KEY,

UrunID INT FOREIGN KEY REFERENCES Urun(UrunID),

Adet INT,

SatisTarihi DATETIME

);

Sample Data:

-- Urun

(1, 'Laptop', 15000.00), (2, 'Mouse', 250.00), (3, 'Klavye', 450.00) -- Satis

(1, 1, 2, '2024-01-10'), (2, 2, 5, '2024-01-15'), (3, 1, 1, '2024-02-20'), (4, 3, 3, '2024-03-05'), (5, 2, 7, '2024-03-25'), (6, 3, 2, '2024-04-12')

metin, ekran görüntüsü, yazılım, ekran, görüntüleme içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Tasks:

1. Write a query that returns, per year and per product, the total sales amount (Fiyat \* Adet) and total quantity.

metin, ekran görüntüsü, yazılım, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

1. For each year, identify the product with the highest sales amount.

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

1. Write a query to list products that were never sold.

metin, yazı tipi, sayı, numara, ekran görüntüsü içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Note: Use MSSQL syntax.