MSSQL (Microsoft SQL Server):

1. **Temel SQL Bilgisi:**

- Temel SQL sorgularını açıklayın:

SELECT: Belirli sütunları seçmek için kullanılır. Örnek: SELECT column1, column2 FROM TableName;

FROM: Hangi tablodan veri alınacağını belirtir. Örnek: SELECT column1, column2 FROM TableName;

WHERE: Belirli bir koşulu sağlayan satırları seçmek için kullanılır. Örnek: SELECT column1, column2 FROM TableName WHERE condition;

ORDER BY: Sonuçları belirli bir sırayla getirmek için kullanılır. Örnek: SELECT column1, column2 FROM TableName ORDER BY column1 ASC;

GROUP BY: Belirli bir sütuna göre gruplama yapmak için kullanılır. Örnek: SELECT column1, COUNT(*) FROM TableName GROUP BY column1;

- INNER JOIN ve LEFT JOIN arasındaki farkı açıklayın:

INNER JOIN iki tablo arasındaki ortak satırları birleştirir. Ortak olan satırlar dışında kalanlar görünmez. LEFT JOIN ise sol tablonun tüm satırlarını ve eşleşen sağ tablo satırlarını getirir. Eğer eşleşme yoksa, sağ tablo için NULL değerler getirilir.

- Aggregate fonksiyonları (COUNT, SUM, AVG, MAX, MIN) açıklayın:

Count: Belirli bir sütundaki toplam satır sayısını döner.

Sum: Belirli bir sütundaki sayısal değerlerin toplamını döner.

Avg: Belirli bir sütundaki sayısal değerlerin ortalamasını döner.

Max: en büyük değeri döner

Min: en küçük değeri döner

2. **Veritabanı Tasarımı:**

- Bir veritabanı tasarım sürecini açıklayın:

Bir veritabanı tasaarımı sürecinde ilk olarak gereksinimler analiz edilir ardından ilişkiler modellenerek entityler tanımlanır. Bu aşamalardan sonra normalizasyon aşamasında veritabanındaki tekrarlanma durumlarından kaçılır ve veri bütünlüğü sağlanır. Tablolar tasarlanır fieldlar belirlenir ardından daha önce belirlenen ilişkiler çerçevesinde indexler belirlenir.

- Normalleştirme nedir? İlk, ikinci ve üçüncü normal formu açıklayın:

Normalizasyon veritabanını optimize etmek için kullanılan bir yöntemler bütünüdür.

1NF: her hücre tek değer içermeli

Table 1: 1NF uygun değil

Id	Ad	Dersler
1	Veysel	Mat, Fizik
2	Ahmet	Kimya, biyoloji

Table 2: 1NF Uygun

Id	AD	Ders
1	Veysel	Mat
1	Veysel	Fizik
2	Ahmet	Kimya
2	Ahmet	Biyoloji

2NF: tablo 1NF uygun olmalı, primary key belirlenmeli, kısmi bağımlılıklar giderilmeli(foreign keyler ile)

Table 3: 2NF uygun

DersId	DersAdı	OgrenciId	Adı
1	Mat	1	Veysel
2	Fizik	1	Veysel
3	Kimya	2	Ahmet
4	Biyoloji	2	Ahmet

3NF: tablo 2NF uygun olmalı, bağımlılıklar giderilmeli(tablolar bölünmeli)

Table 4: 3NF uygun

DersID	DersAd
1	Mat
2	Fizik
3	Kimya
4	Biyoloji

Table 5: 3NF uygun

Id	Ad
1	Veysel
2	Ahmet

3. **İndeksleme:**

- İndeksleme nedir? Hangi durumlarda kullanılır?

İndeksleme, genel olarak bir koleksiyon veya veri kümesindeki öğeleri hızlı bir şekilde bulmayı kolaylaştırmak için kullanılan bir yöntemdir. Bu süreç, veri kümesinin belirli özelliklere göre düzenlenmesini ve organize edilmesini içerir, böylece arama veya erişim işlemleri daha hızlı gerçekleştirilebilir.

İndeksleme; Veritabanları, Arama Motorları, Dosya Sistemleri, Büyük Veri Analizi, Programlama Dilleri ve Dizinleri alanlarında yaygın olarak kullanılır. İndeksleme, genellikle büyük veri kümeleriyle çalışan uygulamalarda performansı artırmak için önemlidir. Ancak, indeksleme işlemi aynı zamanda bellek ve depolama kullanımını artırabilir, bu nedenle dikkatli bir şekilde kullanılmalıdır. İndeksleme, veri güncellendiğinde veya eklendiğinde güncellenmesi gereken bir yapı olduğu için yönetimi önemlidir.

- Clustered ve non-clustered index arasındaki fark nedir?

Clustered ve non-clustered indeksler, veritabanlarında verileri düzenlemek ve erişimini hızlandırmak amacıyla kullanılan iki temel indeks türüdür. İşlevsellik ve etkileri açısından önemli farklılıklar vardır.

Clustered Index (Küme İndeksi):

Clustered indeks, tablonun fiziksel sıralamasını değiştirir ve tablonun kendisiyle aynı düzende saklanır.

Bir tabloda sadece bir adet clustered indeks bulunabilir, çünkü veriler tablo düzenine göre sıralanmıştır.

Veriler fiziksel olarak indeks sırasına göre düzenlendiği için, sıklıkla kullanılan sorguların performansını artırabilir. Clustered indeks, tablo üzerindeki gerçek düzeni temsil eder ve bir sıralama mekanizmasıdır.

Non-Clustered Index (Küme Dışı İndeks):

Non-clustered indeks, veritabanında tablo sıralamasını değiştirmez ve ayrı bir indeks yapısı oluşturur. Bir tabloda birden fazla non-clustered indeks bulunabilir. Veri tablosu ve non-clustered indeks arasında ayrı bir yapı olduğu için, sorgular genellikle bir indeksle yapılan aramalara dayanır ve veri tablosu üzerindeki fiziksel düzeni temsil etmez. Veriler, non-clustered indeksin belirlediği sıralamaya göre değil, tablonun orijinal düzenine göre depolanır.

Farklar:

Fiziksel Düzen: Clustered indeks, veri tablosu ile aynı fiziksel düzeni paylaşır. Non-clustered indeks, veri tablosu ile farklı bir fiziksel düzeni temsil eder. Performans: Clustered indeks, sıklıkla kullanılan sorguların performansını artırabilir, çünkü veriler fiziksel olarak sıralıdır. Non-clustered indeks, sorguların performansını artırabilir ancak genellikle biraz daha düşük performansa sahiptir, çünkü veri tablosu ile farklı bir düzeni temsil eder.

Indeks Sayısı: Bir tabloda yalnızca bir adet clustered indeks bulunabilir. Bir tabloda birden fazla non-clustered indeks bulunabilir.

Veri Güncelleme İşlemleri: Clustered indeks, tablonun fiziksel düzenini değiştirdiği için veri güncellemeleri maliyetli olabilir. Non-clustered indeks, veri tablosu üzerinde doğrudan bir etkisi olmadığı için veri güncellemeleri genellikle daha hızlı gerçekleşir. Hangi indeks türünün kullanılacağı, kullanım senaryosuna, sık kullanılan sorgulara ve veritabanı tasarımı hedeflerine bağlı olarak değişir.

4. **Stored Procedures ve Triggers:**

- Stored procedure nedir? Avantajları nelerdir?

Stored Procedure (Saklanmış Prosedür), bir veritabanı yönetim sistemi (DBMS) içinde önceden derlenmiş ve saklanmış bir SQL kodu bloğudur. Bu prosedür, adlandırılmış bir şekilde çağrılabilir ve genellikle bir dizi SQL ifadesi içerir. Stored procedure'lar, veritabanında belirli görevleri gerçekleştirmek için kullanılır ve genellikle uygulama geliştirme ve veritabanı yönetimi süreçlerinde yaygın olarak kullanılır.

Stored procedure'ların avantajları şunlardır:

Performans Artışı: Stored procedure'lar bir kez derlendiğinde, veritabanı tarafından önbelleğe alınır ve tekrar tekrar çağrıldığında daha hızlı çalışır. Bu, sorgu maliyetini azaltarak performans artışı sağlar.

Araştırma ve Bakım Kolaylığı: Stored procedure'lar, bir kez yazıldıkları ve veritabanına yüklendikleri için, uygulama kodu içinde tekrar tekrar yazma ihtiyacını azaltır. Bu da kodun bakımını ve güncellemesini kolaylaştırır.

Güvenlik Kontrolü: Stored procedure'lar, veritabanı üzerinde belirli işlemleri gerçekleştirmek için kullanılabilir ve bu nedenle güvenlik avantajı sağlar. Kullanıcıların sadece belirli stored procedure'ları çalıştırma yetkisi olabilir.

Modülerlik ve Yeniden Kullanılabilirlik: Stored procedure'lar, modüler bir yapıya sahiptir ve birbiriyle ilişkili işlemleri bir araya getirerek daha düzenli ve okunabilir kod yazmayı sağlar. Aynı stored procedure farklı yerlerde çağrılabilir, bu da kodun yeniden kullanılabilir olmasını sağlar.

Optimizasyon ve İyileştirme: Stored procedure'lar, veritabanı yöneticilerine performans optimizasyonu ve indeksleme stratejileri uygulama imkanı tanır. Bu, veritabanı performansını artırmak için önemlidir.

Transaction Yönetimi: Stored procedure'lar, birden çok SQL ifadesini içerebilir ve bu ifadeleri bir işlem içinde gruplandırabilir. Bu sayede, işlemler atomik (tüm veya hiçbiri), tutarlı (sistem durumunu koruyan) ve izole (diğer işlemlerden bağımsız) bir şekilde gerçekleştirilebilir.

Azalan Ağ Trafik**:** Stored procedure'lar, uygulama tarafında değil, veritabanı tarafında çalıştıkları için, uygulama ve veritabanı arasındaki ağ trafiğini azaltır. Bu, özellikle büyük veri setleri üzerinde çalışan uygulamalarda performans açısından önemlidir.

Stored procedure'lar, genellikle büyük ve karmaşık veritabanlarıyla çalışan uygulamalarda kullanılır ve veritabanı yönetimi, güvenlik ve performans açısından bir dizi avantaj sunar.

- Trigger nedir ve ne zaman kullanılır?

Trigger (tetikleyici), bir veritabanı yönetim sistemi içinde belirli bir olayın meydana gelmesini bekleyen ve bu olay gerçekleştiğinde otomatik olarak çalışan özel bir SQL kodu parçasıdır. Trigger'lar, genellikle belirli bir tablada yapılan veri değişikliklerini (ekleme, güncelleme, silme) takip etmek ve bu değişikliklere tepki vermek için kullanılır. Bir trigger, genellikle INSERT, UPDATE veya DELETE gibi belirli bir tablodaki bir veri değişikliği olayına bağlı olarak tetiklenir. Trigger'lar, veritabanı işlemlerini otomatikleştirmek, iş kurallarını uygulamak ve veritabanı bütünlüğünü korumak için kullanılır.

Bazı temel trigger türleri şunlardır:

BEFORE Triggers (Önce Tetikleyiciler):

Veri ekleneceği, güncelleneceği veya silineceği önce çalışan trigger'lardır.

Veritabanında bir değişiklik yapılmadan önce belirli bir işlemi gerçekleştirmek için kullanılır. Örneğin, değer kontrolü veya veri düzeltme işlemleri.

AFTER Triggers (Sonra Tetikleyiciler):

Veri eklendiği, güncellendiği veya silindiği zaman çalışan trigger'lardır.

Veritabanında bir değişiklik yapıldıktan sonra belirli bir işlemi gerçekleştirmek için kullanılır. Örneğin, başka bir tabloya veri eklemek veya bir log kaydı oluşturmak.

INSTEAD OF Triggers (Bunun Yerine Tetikleyiciler):

Veri ekleme, güncelleme veya silme işlemi yapılacağı zaman, normal işlem yerine bu trigger çalıştırılır.

Özellikle veritabanı tablosundaki belirli bir olayın standart işlevselliğini değiştirmek veya geçersiz kılmak için kullanılır.

Trigger'lar aşağıdaki durumlar için kullanılabilir:

Veri Bütünlüğü Kontrolü: Belirli koşulları karşılayan veya karşılamayan veri girişlerini önlemek için trigger'lar kullanılabilir.

Loglama ve İzleme: Veritabanındaki belirli olayları izlemek ve bu olayları bir log dosyasına kaydetmek için trigger'lar kullanılabilir.

Kaskad Silme veya Güncelleme İşlemleri: Bir tablodaki veri değiştiğinde, başka tablolardaki ilgili verileri otomatik olarak güncellemek veya silmek için trigger'lar kullanılabilir.

Denetim ve Güvenlik: Belirli bir olayın gerçekleşmesine bağlı olarak belirli kullanıcıların veya rollerin erişimini kontrol etmek için trigger'lar kullanılabilir.

Tarih ve Zaman İşlemleri: Belirli bir tarih ve saat aralığına bağlı olarak belirli işlemleri tetiklemek için trigger'lar kullanılabilir.

Trigger'lar, veritabanı uygulamalarının karmaşıklığını artırabilir ve dikkatlice tasarlanmalıdır, aksi takdirde performans ve bütünlük sorunlarına neden olabilirler. İyi bir veritabanı tasarımı ve trigger kullanımı, veritabanı yöneticilerine ve geliştiricilere büyük bir esneklik sağlayabilir.

5. **Transaction (İşlem):**

- Transaction?

SQL'de transaction, bir veya daha fazla SQL ifadesinin gruplandığı ve bu ifadelerin birlikte çalıştığı bir mantıksal birimdir. İşlemler, veritabanında bir dizi değişiklik yapılırken bu değişikliklerin tutarlı bir şekilde uygulanmasını sağlamak için kullanılır

SQL'de işlemler genellikle şu anahtar kelimelerle belirtilir:

BEGIN TRANSACTION: İşlemi başlatır.

COMMIT: İşlemi başarıyla tamamlar ve yaptığı değişiklikleri veritabanına kalıcı hale getirir.

ROLLBACK: İşlemi geri alır ve yaptığı değişiklikleri geri alır. Hata durumlarında veya istenmeyen durumlarda kullanılır.

- ACID özellikleri nelerdir?

ACID, veritabanı işlemlerinin güvenilirliğini ve bütünlüğünü sağlamak amacıyla kullanılan dört temel özelliği temsil eden bir akronymdir. ACID özellikleri şunlardır:

Atomiklik (Atomicity): Bir işlem (transaction), içinde bulunan tüm SQL ifadelerini başarıyla tamamlar veya hiçbirini tamamlamaz. Eğer bir işlem içinde bir hata oluşursa, işlem geri alınır ve veritabanı başlangıç durumuna döner. Atomiklik, işlemlerin kısmi bir şekilde gerçekleşmesini önler.

Tutarlık (Consistency): İşlem başladığında veritabanındaki veriler bir tutarlılık durumundan diğerine geçirilir. İşlem başarıyla tamamlandığında, veritabanı kısıtlamalara (örneğin, benzersizlik kısıtlamaları) uygun bir durumda olmalıdır. Tutarlık, veritabanının bütünlüğünü korur.

İzolasyon (Isolation): İki eş zamanlı işlem birbirinden bağımsız olarak çalışır ve bir işlem diğer işlemlerin yaptığı değişiklikleri görmemelidir. İzolasyon, bir işlemin diğer işlemlerle etkileşimde bulunmamasını sağlar ve böylece eş zamanlılık sorunlarını önler.

Dayanıklılık (Durability): Bir işlem başarıyla tamamlandığında, yapılan değişiklikler kalıcı hale gelir ve veritabanı sistemini yeniden başlatsa veya bir hata oluşsa bile bu değişiklikler korunur. Dayanıklılık, veritabanındaki verilerin kalıcı bir şekilde saklanmasını garanti eder.

ACID özellikleri, veritabanı işlemlerinin güvenilirliğini ve bütünlüğünü sağlamak için önemlidir. Bu özellikler, özellikle iş kritik uygulamalarda ve finansal sistemlerde, veritabanı işlemlerinin doğru ve güvenilir bir şekilde gerçekleşmesini sağlamak amacıyla kullanılır. ACID uyumluluğu,

veritabanı yöneticileri ve uygulama geliştiricileri için temel bir tasarım prensibi olarak kabul edilir.

RESTful Servisler:

1. **REST Nedir?**

- REST'in açılımını açıklayın.

REST, Representational State Transfer'ın kısaltmasıdır. REST, web tabanlı uygulamalar arasında veri iletimi için kullanılan bir mimari stilini ifade eder. REST, özellikle HTTP protokolü üzerinde çalışan web servisleri ve uygulamaları için popüler bir yaklaşımdır.

- RESTful servislerin temel prensipleri nelerdir?

Temsil Durumu Aktarımı (Representational State Transfer): Verilerin ve kaynakların temsil durumu, bir kaynağın mevcut durumunu temsil eden verilerin bir gösterimidir. İstemci ve sunucu arasındaki iletişim, kaynakların temsil durumunu taşıyan mesajlar üzerinden gerçekleşir.

İstemci-Sunucu Mimarisi (Client-Server Architecture): İstemci ve sunucu, bağımsız bileşenlerdir ve birbirinden ayrıdır. İstemci, kullanıcı arayüzü ve kullanıcı etkileşimi ile ilgili görevleri gerçekleştirir. Sunucu, kaynakları yönetir ve işlemleri gerçekleştirir. Bu ayrım, ölçeklenebilirlik ve bağımsız geliştirme avantajları sağlar.

Önbellekleme (Cacheability): Sunucudan alınan cevaplar, istemci tarafından önbelleğe alınabilir. Bu sayede, aynı veriye tekrar tekrar erişim sağlanarak performans artırılabilir.

Katmanlı Mimariler (Layered Architectures): REST, katmanlı bir mimariyi destekler. Her katman, belirli bir sorumluluk alanına sahiptir ve işlevselliği daha iyi organize etmek ve güvenliği artırmak için kullanılır.

Tekrar Kullanılabilirlik (Statelessness): İstemci ve sunucu arasındaki her istek, yeterli bilgi içermelidir ve önceki isteklerle bağlantılı olmamalıdır. Sunucu, her isteği tek başına işleyebilmelidir. Bu, sistemdeki durumun (state) istemci ve sunucu arasında paylaşılmamasını ifade eder.

REST, genellikle HTTP protokolü üzerinde çalışır, ancak bu zorunlu değildir. Kaynakların tanımlanması, temsil durumlarının kullanımı ve HTTP metodları (GET, POST, PUT, DELETE gibi) REST'in temel bileşenlerindendir. RESTful web servisler, bu prensiplere uygun olarak tasarlanan web servisleridir ve genellikle HTTP üzerinden erişilebilir.

2. **HTTP Metotları:**

- GET ve POST metotları arasındaki farkı açıklayın.

GET, veriyi URL üzerinden gönderirken ve tarayıcıya bir şeyi talep ederken kullanılır. POST, veriyi güvenli bir şekilde (URL'de görünmeyecek şekilde) gönderirken ve genellikle sunucuya bir şeyi gönderirken kullanılır. Form verilerini göndermek, dosya yüklemek gibi durumlar için uygundur.

- PUT ve PATCH arasındaki farkı açıklayın.

PUT ve PATCH, HTTP protokolünde kaynakların güncellenmesi için kullanılan iki farklı metottur. Bu iki metot arasındaki temel fark, nasıl çalıştıkları ve kaynakların ne kadarını güncelledikleri ile ilgilidir. Eğer bir kaynağı tamamen değiştirmek isteniyorsa ve istemci tarafından gönderilen veri, kaynağın tamamını temsil ediyorsa PUT kullanılabilir. Eğer bir kaynağın sadece belirli alanlarını güncellemek isteniyorsa ve gönderilen veri sadece güncellenmek istenen alanları içeriyorsa PATCH kullanılabilir.

Genel olarak, güncelleme işlemi ne kadar spesifik ve kısıtlı olursa, PATCH kullanmak daha uygun olabilir. Ancak, özellikle kaynağın tamamını güncellemek gerekiyorsa PUT tercih edilebilir.

3. **RESTful API Tasarımı:**

- Bir kaynağı temsil eden bir URL nasıl tasarlanır?

Bir kaynağı temsil eden bir URL tasarlanırken, tasarımın tutarlı, anlamlı ve kullanımı kolay olması önemlidir. RESTful API tasarımında, URL'ler genellikle kaynakları temsil eder ve bu nedenle URL tasarımı, uygulamanın mantığına ve kullanıcı beklentilerine uygun olmalıdır.

- HATEOAS (Hypermedia As The Engine Of Application State) nedir?

HATEOAS, "Hypermedia As The Engine Of Application State" teriminin kısaltmasıdır. Bu kavram, REST (Representational State Transfer) mimarisinde bir ilkedir ve web servislerinin tasarımını etkileyen önemli bir prensiptir. HATEOAS, özellikle RESTful API'lerde kullanılarak uygulamaların durum yönetimini ve etkileşimini yönetme konusunda önemli bir rol oynar. HATEOAS'in temel fikri, bir uygulamanın durumunu yönetme ve etkileşimde bulunma konusundaki bilgilerin sunucu tarafından sağlanmasıdır. Bu, istemci uygulamanın, sunucudan alınan temsil durumu içinde bulunan linkler ve bilgiler üzerinden uygulama durumunu anlamasını sağlar. İstemci, sunucudan aldığı bu bilgilerle bir sonraki geçerli adımı belirleyebilir ve bu adımı gerçekleştirmek için gerekli kaynaklara erişebilir.

HATEOAS'in temel bileşenleri şunlardır:

Durum Temsilcisi (State Representation): Sunucu, istemciye bir kaynağın durumunu temsil eden bir temsil (representation) gönderir. Bu temsil, JSON, XML veya başka bir formatta olabilir. Temsil durumu, kaynağın mevcut durumunu ve ilgili linkleri içerir.

Linkler (Links): Temsil durumu içinde bulunan linkler, istemcinin belirli aksiyonları gerçekleştirmek veya ilgili kaynaklara erişmek için kullanabileceği bağlantıları temsil eder. Linkler, bir kaynağın geçerli durumuna bağlı olarak dinamik olarak değişebilir.

Aksiyonlar ve İzinler (Actions and Permissions): Linkler genellikle belirli aksiyonları veya istemcinin yapabileceği belirli işlemleri temsil eder. Bu, istemcinin bir kaynağın belirli durumlarına uygun olarak aksiyonlarını belirlemesine yardımcı olur.

HATEOAS'in avantajları şunlardır:

Dinamik ve Esnek İnteraksiyon: İstemci, linkleri kullanarak dinamik olarak uygulama durumunu keşfeder ve bu duruma uygun olarak etkileşimde bulunabilir.

İstemci ve Sunucu Bağımsızlığı: İstemci ve sunucu, birbirlerinden bağımsız olarak geliştirilebilir ve değiştirilebilir. İstemci, linklere dayalı olarak sunucuyla etkileşimde bulunabilir.

Evrim: Sunucu tarafında yapılan değişiklikler, temsil durumu ve linkler aracılığıyla istemcilere iletilerek, sistemde evrilebilirlik sağlanabilir.

HATEOAS, RESTful API'lerde uygulama durumu yönetimini iyileştirerek, daha esnek, ölçeklenebilir ve sürdürülebilir uygulamaların geliştirilmesine katkı sağlar.

4. **HTTP Durum Kodları:**

- En çok kullanılan HTTP durum kodları nelerdir? (200, 201, 400, 401, 404, 500, vb.)

HTTP durum kodları, bir HTTP isteğinin sonucunu tanımlayan üç haneli sayılardır. İşte en çok kullanılan bazı HTTP durum kodları ve kısa açıklamaları:

200 OK: Başarılı bir GET isteği veya başarılı bir PUT veya POST isteğinin sonucu olarak kullanılır.

201 Created: Yeni bir kaynak başarıyla oluşturulduğunda veya bir PUT veya POST isteği ile bir kaynak güncellendiğinde kullanılır.

204 No Content: Bir isteğin başarıyla işlenmesi durumunda, ancak dönecek bir içerik olmadığında kullanılır.

400 Bad Request: İstek, sunucu tarafından anlaşılamayacak bir biçimde yapıldığında veya geçersiz veri gönderildiğinde kullanılır.

401 Unauthorized: İstemcinin kimlik doğrulamasının gerektiği ve doğrulanmadığı durumlarda kullanılır.

403 Forbidden: İstemcinin, kaynağa erişim yetkisinin olmadığı durumlarda kullanılır.

404 Not Found: İstenen kaynak bulunamadığında kullanılır. Yani, URI geçersizse veya kaynak mevcut değilse.

405 Method Not Allowed: İstemci tarafından gönderilen HTTP metodu, kaynak üzerinde desteklenmiyorsa kullanılır.

409 Conflict: İstek, kaynak üzerinde bir çatışma oluşturduğunda, örneğin aynı anda iki kullanıcı tarafından yapılan bir güncelleme çakışması durumunda kullanılır.

500 Internal Server Error: Sunucu tarafında bir genel hata oluştuğunda veya sunucu isteği işleyemiyorsa kullanılır.

503 Service Unavailable: Sunucu geçici olarak hizmet veremiyor veya aşırı yük altında olduğunda kullanılır.

Bu durum kodları, HTTP protokolü üzerindeki temel durumları temsil eder. Ancak, HTTP protokolü daha pek çok durum kodunu destekler ve her durum kodu belirli bir senaryoyu temsil eder. Geliştiriciler, durum kodları aracılığıyla uygulamalarının davranışını daha iyi anlayabilir ve kullanıcılarına uygun mesajlar gönderebilirler.

SOAP Servisler:

1. **SOAP Nedir?**

- SOAP'ın açılımını açıklayın.

SOAP, "Simple Object Access Protocol"ın (Basit Nesne Erişim Protokolü) kısaltmasıdır. SOAP, bilgisayarlar arasında iletişim kurmak ve web servisleri arasında veri alışverişi sağlamak için kullanılan bir protokoldür. Temelde XML tabanlı bir iletişim protokolüdür ve genellikle HTTP, HTTPS, SMTP gibi iletişim protokollerini kullanarak veri transferini gerçekleştirir. SOAP, XML formatında metin tabanlı bir mesaj formatını kullanır ve genellikle web servislerinin tanımlanmasında ve iletişiminde kullanılır.

- SOAP servislerin avantajları ve dezavantajları nelerdir?

SOAP'un Avantajları:

Platform ve Dil Bağımsızlığı: SOAP, farklı platformlar ve programlama dilleri arasında bağımsız çalışabilir. Bu, heterojen sistemlerde (farklı teknolojilerin bir arada kullanıldığı sistemler) kullanımını kolaylaştırır.

ACID Uyumlu: SOAP protokolü, ACID özelliklerini destekler. Bu, veritabanı işlemlerinin güvenilir ve tutarlı bir şekilde gerçekleştirilmesini sağlar.

Gelişmiş Güvenlik: SOAP, WS-Security gibi geniş güvenlik standartlarını destekler. Bu, veri şifreleme, kimlik doğrulama ve yetkilendirme gibi güvenlik önlemlerini uygulama olanağı sağlar.

Dahili Hata Yönetimi: SOAP, hata durumlarını işlemek için Fault elemanını kullanır. Bu, uygulama tarafında oluşan hataları belirtmek ve işlemi başarılı bir şekilde tamamlayamayan durumları ele almak için kullanılır.

Formalite ve Standartlar: SOAP, WSDL (Web Services Description Language) ve UDDI (Universal Description, Discovery, and Integration) gibi standartları destekleyerek, web servislerinin formal tanımını ve keşfini sağlar.

SOAP'un Dezavantajları:

Karmaşıklık: SOAP mesajları genellikle XML tabanlıdır ve bu da mesajların boyutunu artırabilir ve okunurluğunu azaltabilir. Bu durum, veri transferinde gereksiz yük ve karmaşıklık oluşturabilir.

Performans: SOAP protokolü, XML tabanlı mesajlar kullanarak, REST gibi daha hafif protokollere kıyasla daha yüksek bir veri transfer maliyetine sahip olabilir. Bu durum, performans konusunda REST'e kıyasla bir dezavantaj oluşturabilir.

Daha Az Tarayıcı Desteği: SOAP web servisleri, genellikle tarayıcı tabanlı uygulamalarda değil, daha çok sunucu-tabanlı uygulamalarda kullanılır. Bu, REST'in tarayıcılarla daha iyi entegrasyon sağlama avantajına sahip olmasını sağlar.

Esneklik: SOAP, REST'e kıyasla daha katı bir yapıya sahiptir. Bu, özellikle hafif ve esnek bir iletişim protokolü gerektiren modern web uygulamaları için dezavantaj olabilir.

Keşif Zorlukları: SOAP web servislerini keşfetmek için kullanılan UDDI protokolü, pek çok uygulama tarafından benimsenmemiştir ve bu nedenle servislerin keşfi konusunda zorluklar ortaya çıkabilir.

Genel olarak, SOAP'un güvenlik ve standartlar konusundaki zengin özellikleri, kurumsal düzeydeki uygulamalarda ve özellikle veri bütünlüğü ve güvenliği kritik olan sistemlerde tercih edilmesini sağlar. Ancak, hafiflik ve basitlik arayan modern web uygulamalarında REST gibi protokoller daha yaygın olarak tercih edilmektedir.

2. **WSDL **

- WSDL nedir? Ne işe yarar?

WSDL, "Web Services Description Language"ın kısaltmasıdır ve web servislerinin tanımlanması ve bu servislerle etkileşim kurma süreçlerini açıklamak için kullanılan bir XML tabanlı metin formatıdır. WSDL, SOAP tabanlı web servislerinin, istemcilerin ve diğer servislerin bu servislerle nasıl etkileşimde bulunacaklarını tanımlayan bir sözleşme (contract) sağlar.

WSDL'nin temel işlevleri şunlardır:

Servis Tanımı: WSDL, bir web servisinin genel tanımını sağlar. Bu, servisin adı, konumu (URL), kullanılan protokoller (genellikle SOAP), ve servisin sunduğu operasyonlar gibi temel bilgileri içerir.

Mesaj Yapısı: WSDL, web servisi tarafından desteklenen operasyonlar için giriş ve çıkış mesajlarının yapısını tanımlar. Bu, mesajlardaki parametreleri, veri türlerini ve iletilen veri yapısını içerir.

Bağlayıcılar ve Protokoller: WSDL belgesi, web servisi ile iletişim kurmak için kullanılan bağlayıcıları (binding) ve iletişim protokollerini tanımlar. Genellikle bu bağlayıcılar, SOAP protokolü üzerinde çalışan web servislerini temsil eder.

Operasyonlar ve İşlemler: WSDL belgesi, bir web servisinin sunmuş olduğu operasyonları ve bu operasyonların imzalarını (parametreleri ve dönüş değerlerini) tanımlar.

WSDL belgesi, web servislerini kullanacak olan istemcilerin ve diğer servislerin, servis sağlayıcının sunduğu hizmetleri doğru bir şekilde anlamalarına yardımcı olur. Bu belge, servisin nasıl çağrılacağı, hangi parametrelerin kullanılacağı, ve servisten beklenen yanıtların yapısı hakkında detaylı bilgiler içerir. WSDL, genellikle web servislerinin tanımlandığı, kullanıldığı ve keşfedildiği kurumsal düzeydeki uygulamalarda önemli bir rol oynar. Özellikle SOA (Service-Oriented Architecture) prensiplerine uyan ve farklı sistemler arasında etkileşim sağlamak isteyen büyük ve karmaşık sistemlerde WSDL kullanımı yaygındır.

3. **SOAP ve REST Karşılaştırması:**

- SOAP ve REST arasındaki temel farklar nelerdir?

SOAP (Simple Object Access Protocol) ve REST (Representational State Transfer), web servisleri ile iletişimde kullanılan iki farklı mimari yaklaşımıdır. İşte SOAP ve REST arasındaki temel farklar:

Protokol: SOAP, kendi protokolüne (HTTP, SMTP, TCP gibi) sahip bir iletişim protokolüdür. REST ise genellikle HTTP protokolü üzerinde çalışır, ancak HTTP'ye bağımlı değildir ve başka protokollerle de kullanılabilir.

Veri Formatı: SOAP, genellikle XML tabanlı mesaj formatını kullanır. REST ise JSON, XML, HTML veya metin gibi çeşitli veri formatlarını destekler.

Karmaşıklık: SOAP, genellikle karmaşık ve ağır bir protokoldür. XML tabanlı mesajlar ve detaylı şemalar, karmaşıklığı artırabilir. REST ise daha basit ve hafif bir yapıya sahiptir, genellikle JSON formatıyla çalışır ve okunabilirliği artırır.

Statefulness (Durum Bilgisi): SOAP, stateful (durum bilgisi taşıyan) bir yapıya sahip olabilir. Mesajlar arasında durum bilgisi taşıma yeteneği vardır. REST ise stateless (durum bilgisi taşımayan) bir mimari prensibi benimser. Her istek, bağımsız ve kendi içinde anlamlıdır.

Performans: REST, daha hafif veri formatları ve stateless yapısı nedeniyle genellikle daha hızlıdır. SOAP ise XML tabanlı veri formatı ve stateful yapısı nedeniyle daha fazla veri taşır ve daha yavaş olabilir.

Bağlam: SOAP genellikle kurumsal düzeydeki uygulamalarda tercih edilir. Özellikle güvenlik ve bütünlük konularında ihtiyaç duyulan karmaşık senaryolara yöneliktir. REST ise daha hafif ve basit bir yapıya sahiptir ve genellikle web ve mobil uygulamalarda kullanılır.

Standartlar: SOAP, belirli standartlara dayanır ve WS-* standartlarını kullanabilir. REST ise bir standarda bağlı olmaksızın, HTTP metotları ve URI yapıları üzerine inşa edilen basit bir mimaridir.

Uygulama Alanı: SOAP, genellikle büyük ve karmaşık sistemlerde, özellikle kurumsal uygulamalarda tercih edilir. REST ise özellikle web ve mobil uygulamalarda kullanılmak üzere daha uygun bir mimari sunar.

Hangi mimarinin kullanılacağı, projenin ihtiyaçlarına, güvenlik gereksinimlerine, performans beklentilerine ve mevcut altyapıya bağlı olarak belirlenir.

- Hangi durumlarda SOAP tercih edilir?

REST, daha basit ve genellikle ölçeklenebilir ve hızlı uygulamalar için tercih edilirken, SOAP, karmaşık entegrasyon ve güvenlik gereksinimleri olan kurumsal uygulamalarda daha yaygındır.

4. **XML ve JSON:**

- SOAP mesajları genellikle XML formatında olurken, RESTful servisler JSON kullanır. Bu iki format arasındaki farkları açıklayın.

SOAP mesajları genellikle XML formatında olurken, RESTful servisler JSON kullanma eğilimindedir. Her iki format da veri taşıma amacı güder, ancak XML ve JSON arasında bazı temel farklar bulunmaktadır. İşte XML ve JSON arasındaki bazı farklar:

Okunabilirlik:

XML: XML, insanlar tarafından okunabilir ancak daha fazla veri taşır. Etiket tabanlı bir formata sahiptir, bu da belgeleri büyük ve karmaşık hale getirebilir.

JSON: JSON, daha kompakt ve okunabilir bir formata sahiptir. Anahtar-değer çiftleri ve diziler kullanır, bu nedenle daha sade ve kısa belgeler oluşturabilir.

Veri Taşıma Etkinliği:

XML: XML genellikle daha fazla veri taşır, bu da daha büyük veri transferleri ve daha geniş bant genişlikleri gerektirebilir.

JSON: JSON, daha hafif bir formata sahiptir, bu nedenle daha az veri taşır ve genellikle daha etkin bir şekilde kullanılır.

Yazım Kolaylığı:

XML: XML belgeleri, etiketler ve açıklamalar nedeniyle daha uzun ve karmaşık olabilir. El yazımı daha zordur.

JSON: JSON, basit bir anahtar-değer yapısına sahiptir ve genellikle daha az sembol içerir. Bu, yazımını kolaylaştırır.

Veri Türleri:

XML: XML, farklı veri tiplerini desteklemek için geniş bir yapı sağlar. Özel veri türleri, şemalar ve isim alanları kullanabilir.

JSON: JSON, temel veri tiplerini (sayılar, metin, boolean, nesneler, diziler, null) destekler ancak daha sınırlıdır. JSON şemaları da XML'e kıyasla daha az karmaşık ve yaygın olarak kullanılan bir standart değildir.

Dil Bağımsızlık:

XML: XML, dil bağımsızdır ve herhangi bir veri yapısını temsil edebilir. Bu nedenle, bir dilden diğerine daha kolay çevrilebilir.

JSON: JSON, dil bağımsızdır ancak daha çok JavaScript nesne yapısına benzer. Diğer dillere çevirme süreci, dilin desteklediği nesne yapısına bağlı olarak değişebilir.

Açıklama (Schema):

XML: XML, genellikle XML Şemaları (XSD) kullanılarak belgelenir ve tanımlanır.

JSON: JSON için açıklama dilinin kullanımı daha az yaygındır ve JSON Şemaları (JSON Schema) gibi standartlar daha az kullanılır.

Her iki formatın da kullanılması gereken durumlar vardır ve seçim, projenin ihtiyaçlarına ve gereksinimlerine bağlıdır. JSON, RESTful servislerle daha yaygın olarak kullanılırken, SOAP'un XML tabanlı yapısı genellikle daha karmaşık sistemler ve entegrasyonlar için tercih edilmektedir.