SQL (İlişkisel Veritabanları) Kullanım Amacı:

- Yapılandırılmış veriler için
- Karmaşık sorgular ve işlemler
- Veri tutarlılığının kritik olduğu durumlar
- ACID prensiplerinin önemli olduğu uygulamalar

Örnekler:

- PostgreSQL
- MySQL
- Oracle
- Microsoft SQL Server

NoSQL (İlişkisel Olmayan Veritabanları) Kullanım Amacı:

- Esnek şema ihtiyacı
- Yüksek ölçeklenebilirlik gereken durumlar
- Büyük veri uygulamaları
- Hızlı okuma/yazma gerektiren sistemler

Örnekler:

- MongoDB (Doküman tabanlı)
- Redis (Key-value)
- Cassandra (Geniş sütun)
- Neo4j (Graf tabanlı)

NewSQL (Modern İlişkisel Veritabanları) Kullanım Amacı:

- SQL'in ACID özelliklerini koruma
- NoSQL'in ölçeklenebilirlik avantajlarını sağlama
- Yüksek performanslı OLTP işlemleri
- Modern dağıtık sistemler

Örnekler:

- CockroachDB
- Google Spanner
- VoltDB
- NuoDB

SQL, NoSQL ve NewSQL, farklı veri yönetimi ihtiyaçlarına cevap veren veritabanı yaklaşımlarıdır. SQL (İlişkisel veritabanları), önceden tanımlanmış şema yapısı, ACID özellikleri ve güçlü veri tutarlılığı ile öne çıkarken, kompleks sorgular için idealdir ancak yatay ölçeklendirme konusunda sınırlıdır.

NoSQL ise şema esnekliği, yüksek performans ve kolay yatay ölçeklenebilirlik sunar, fakat veri tutarlılığından ödün verebilir ve karmaşık sorguları destekleme konusunda SQL kadar yetenekli değildir.

NewSQL ise modern bir yaklaşım olarak, SQL'in veri tutarlılığı ve ACID özelliklerini korurken, NoSQL'in ölçeklenebilirlik ve performans avantajlarını birleştirerek, özellikle yüksek hacimli işlem gerektiren fintech gibi alanlarda tercih edilir. Bu üç yaklaşım, birbirlerinin eksikliklerini tamamlayarak farklı kullanım senaryolarına hizmet eder ve genellikle modern uygulamalarda hibrit olarak kullanılırlar.

NoSQL Veritabanlarları

NoSQL ilişkisel veritabanı yönetim sistemlerinin sunduğu yapılandırılmış ve katı tablo yapılarının ötesine geçmek için tasarlanmış bir veritabanı yaklaşımıdır. NoSQL, esnek veri modelleri, yüksek performans, yatay ölçeklenebilirlik ve geniş çapta kullanılabilirlik sağlayarak modern veri ihtiyaçlarını karşılamak üzere geliştirilmiştir. Özellikle büyük veri, gerçek zamanlı analizler ve dağıtık sistemler gibi kullanım senaryolarında öne çıkar.

NoSQL Sistemlerinin Özellikleri

1. **Yapılandırılmamış ve Yarı Yapılandırılmış Verileri Destekleme:** NoSQL veritabanları, JSON, BSON gibi formatlarda saklanabilen verileri destekler. Bu esneklik, değişken veri yapılarının kolayca işlenmesine olanak tanır.
2. **Yatay Ölçeklenebilirlik:** Veritabanları, birden fazla sunucuya kolayca dağıtılabilir, bu da büyük veri kümelerinin ve yüksek trafikli uygulamaların verimli bir şekilde işlenmesini sağlar.
3. **CAP Teoremi Odaklılık:** Consistency ,Availability ve Partition Tolerance arasında bir denge kurar. Genellikle tutarlılık yerine erişilebilirlik tercih edilir.
4. **BASE Prensibi:** ACID yerine BASE yaklaşımını benimser. Bu, daha esnek ve performans odaklı bir veri yönetimi sağlar.

Avantajları

- **Esneklik**: Katı şemalar gerektirmez, bu da veri yapısında değişiklik yapmayı kolaylaştırır.
- **Hızlı Erişim**: Geleneksel ilişkisel sistemlere kıyasla sorgu işlemlerinde daha hızlıdır.
- **Dağıtık Mimari**: Veritabanı verilerini birden fazla sunucuya dağıtarak hata toleransı sağlar.
- **Büyük Veri için Uygunluk**: Yüksek hacimli veri işlemlerinde etkili performans sunar.

Dezavantajları

- **Tutarlılık Sorunları**: BASE prensibi nedeniyle veriler her zaman tam tutarlı olmayabilir.
- **SQL Desteğinin Olmaması**: Bazı NoSQL veritabanlarında SQL sorgularını kullanmak mümkün değildir.
- **Uygulama Karmaşıklığı**: Veritabanı mimarisine bağlı olarak uygulama katmanında daha fazla iş yükü gerektirebilir.

Esnek veri yönetimi ve ölçeklenebilirlik nedeniyle NoSQL, yüksek kullanıcı trafiğinin olduğu sosyal medya platformlarında ve e ticaret uygulamalarında kullanılır.

PostgreSQL'in NoSQL Desteği

PostgreSQL, bir ilişkisel veritabanı sistemi olmasına rağmen, NoSQL özelliklerini de destekleyerek hibrit bir yapı sunar. Bu, hem geleneksel SQL sorgularının hem de NoSQL veri yönetiminin bir arada kullanılabilmesini sağlar. PostgreSQL'in NoSQL ile ilgili öne çıkan özellikleri şunlardır:

1. **JSON ve JSONB Desteği**:
    - JSON: Yarı yapılandırılmış verilerin saklanmasını sağlar. Ancak veri sorgulama işlemleri sırasında performans sınırlamaları olabilir.
    - JSONB: JSON'un ikili formatta optimize edilmiş hali olup daha hızlı sorgulama ve indeksleme imkanı sunar.
2. **HStore Modülü**:
    - Anahtar-değer çiftlerini saklamak ve sorgulamak için kullanılır. Özellikle basit NoSQL kullanım senaryoları için uygundur.
3. **Full-Text Search ve GIN/GiST İndeksleri**:
    - Yüksek hızlı metin arama işlemleri ve indeksleme yetenekleri sağlar.
4. **Yatay Ölçeklenebilirlik**:
    - PostgreSQL, **Sharding** ve **Partitioning** gibi teknikler aracılığıyla büyük veri kümeleri için ölçeklenebilirlik sunar.

PostgreSQL'de Pipeline ve Materialized Yaklaşımlar

Pipeline Yaklaşımı

Pipeline işleme, verilerin sorgu sırasında bir akış halinde işlemden geçirilmesi prensibine dayanır. Veriler, bir işlem tamamlandıktan hemen sonra bir sonraki aşamaya geçirilir ve tüm işlem boyunca ara sonuçlar bellekten diske yazılmadan tutulur.

Pipeline yaklaşımı, ara sonuçları bellekte tutarak düşük gecikme süresi ve yüksek hız sağlar. Bu, sürekli veri akışını gerektiren sistemlerde performansı artırır. Ayrıca, düşük bellek kullanımı sayesinde sistem kaynaklarını verimli şekilde kullanır. Ancak, pipeline işleme karmaşık sorgularda performans sınırlamaları yaşayabilir ve ara sonuçlar saklanmadığı için tekrar kullanım imkanı sunmaz. Bu yöntem, anlık analiz ve işlem ihtiyaçları için ideal bir çözümdür.

Kullanılan Yöntemler

1- Query Execution Pipeline:

- PostgreSQL, bir sorgunun yürütülmesi sırasında bir sorgu planlayıcı ve yürütme motoru kullanır. Pipeline yaklaşımı, bu iki aşamanın birleşimiyle sağlanır.
- Veriler, işlem düğümleri arasında sıralı olarak işlenir.
- Örnek: Bir JOIN işlemi yapılırken, iki tablodan alınan veriler birleştirilir ve daha sonra filtreleme işlemi yapılır.

2- Lazy Evaluation:

- Pipeline işleme, genellikle tembel değerlendirme yöntemiyle çalışır. Bu, yalnızca gereken verilerin işlendiği anlamına gelir.
- Bu yöntem sayesinde bellek kullanımı minimumda tutulur ve yalnızca ihtiyaç duyulan veriler işlenir.

3- Query Plan Nodes:

- PostgreSQL'de her sorgu bir dizi düğümden (node) oluşur. Her düğüm, bir işlemi (örneğin, tarama, filtreleme, birleştirme) temsil eder.
- Pipeline yaklaşımında, bu düğümler ardışık bir şekilde işlenir ve ara sonuçlar bir sonraki düğüme taşınır.

4- JSONB İşleme:

- JSONB veri türü, pipeline işleme sırasında sıkça kullanılır. JSONB verileri üzerinde yapılan işlemler (örneğin, filtreleme, anahtar değer okuma), PostgreSQL'in GIN indeksleme yöntemiyle optimize edilir.

Materialized Yaklaşımı

Materialized işleme, sorgu sonucunda oluşan ara sonuçların diskte saklanması prensibine dayanır. Bu yöntem, özellikle karmaşık sorguların tekrar tekrar çalıştırılmasını önlemek için kullanılır.

Materialized view desteği sayesinde, büyük veri setleri üzerinde yapılan analizler ve raporlama işlemleri hız kazanır. Bu yöntem, özellikle sık tekrarlanan sorgular için yüksek performans sunar ve ara sonuçların yeniden kullanılabilir olmasını sağlar. Ara sonuçların diskte saklanması, bellek kullanımını azaltırken verilerin kalıcılığını garanti eder. Ancak, bu yöntem yüksek depolama alanı gerektirir ve güncelleme işlemleri sırasında gecikmelere yol açabilir. Ayrıca, verilerin diskten okunup yazılması, işlem sürelerini artırabilir. Materialized işleme, büyük veri analitiği ve uzun vadeli veri saklama gereksinimleri için ideal bir çözümdür.

Kullanılan Yöntemler

1- Materialized View Yapısı:

- Materialized view, sorgu sonuçlarının fiziksel olarak diskte saklanmasını sağlar. PostgreSQL, bu yapı için aşağıdaki süreçleri kullanır:
    - Sorgu yürütülür ve sonuçlar bir tabloya yazılır.
    - Bu tablo, yeniden sorgulanabilir bir veri kaynağı olarak kullanılır.

2- Incremental Refresh:

- PostgreSQL, materialized view'leri tam güncelleme veya artımlı güncelleme yöntemleriyle yenileyebilir.
- Artımlı güncelleme, yalnızca değişen veya yeni eklenen verilerin işlenmesi anlamına gelir. Bu, işlem süresini büyük ölçüde azaltır.

3- Index Destekli Optimizasyon:

- Materialized view üzerinde indeksler oluşturulabilir. Bu indeksler, sorgu süresini daha da azaltır.

4- Concurrency Control:

- Materialized view'ler, PostgreSQL'in MVCC (Multi-Version Concurrency Control) yapısıyla uyumlu bir şekilde çalışır. Bu, bir view üzerindeki işlemlerin diğer işlemlerle çakışmasını önler.

Yusuf Safa Köksal

21011002