Interning’in Gücü

Interner’ın akla gelen ilk kullanım alanı stringlerdir. Interning temelde bir string tablosudur. Bir string vektörü ile hash haritasının eşleştirilmesinden oluşur. Interning, bir stringin veri tabanında olup olmadığını ve varsa vektörde hangi indekste bulunduğunu hızlıca sorgulamak için kullanılır. Internlenmiş bir string interning tablosunda bir indeks olur. Bu kurulumun asıl avantajı program tarafından kullanılan bellek miktarını azaltmasıdır. Çünkü tam sayı bir indeks bellekte bütün bir stringden daha az yer kaplar. Çok fazla tekrarlanan stringlerin olduğu bir durumda bu yöntem özellikle etkilidir. Neyse ki interning tekniği sadece stringlere özgü bir teknik değildir. Bir vektör ve bir hash haritasına konabilecek her veriyle bu teknik çalışabilir. Bu durumda istediğimiz veri tipiyle çalışmasını sağlamak amacıyla generics kullanabiliriz. Veri setinde tekrarlanan veri sayısına göre indeks veri tipini bellekte daha az yer tutacak bir veri tipi belirleyerek de bellek kullanımında azaltım yapılabilir. Veri setinde bulunan bazı vektörlerin veri tipi olarak aynı ama sıralanış olarak farklı olduğu durumlar bulunabilir. Ama Rust’da vektörler semantik olarak sıralıdırlar. Bu durumda bu vektörlerin aynı veri tiplerine sahipse ve sırası farklıysa vektörler aynı değerleri tutsa bile semantik olarak aynı olsalar da hashing ve equality olarak farklı olacaklardır. Bu durumu çözmenin en kolay yolu bu vektörleri sıralamaktır. Kavramsal olarak, interning basit bir tekniktir, ancak birçok farklı seçim ve optimizasyona yol açabilir. Genellikle stringlere uygulanır, ancak pratikte gerçekten fark yaratan şey, her türlü veri yapısının geniş kapsamlı olarak interning işlemiyle optimize edilebilmesidir. Ancak interning yönteminin bazı sınırlamaları vardır. Interning yöntemiyle oluşturulan bir veri setine sadece ekleme yapılabilir, nesneler değiştirilemez veya silinemez. Bununla birlikte serileştirilmiş form, rastgele indeks erişimini desteklemez, bu yüzden tüm veri tabanını belleğe yüklemek gerekir. Son olarak yeni bir nesneye interning yaparken senkronizasyon olmadan indeksin artırılmasını gerektirir. Eklenmesi nispeten basit olabilecek bir özellik ise kademeli (incremental) güncellemeleri desteklemektir. Interning tabloları yalnızca ekleme yapacak şekilde tasarlandığından ve indeksler artımlı olduğu için, veri tabanının bir anlık görüntüsünü (snapshot) alabilir ve daha sonra sadece bu anlık görüntüden sonra eklenen yeni nesneleri içeren bir fark (diff) oluşturabilirsiniz. Bu yöntemle birden fazla okuyucu düğümü (reader node) oluşturulabilir ve sonuç olarak çoğaltılmış (replicated) bir veritabanı elde edilebilir. Bu durumda, yazıcı düğüm belirli aralıklarla okuyuculara kademeli güncellemeleri iletebilir. String interning'in avantajları arasında bellek kullanımının azalması, metin karşılaştırmalarının hızlanması ve genel uygulama performansının artması yer almaktadır. Ancak, bu teknik aynı zamanda bazı dezavantajlara da sahiptir. Örneğin, interning yapısının sürekli olarak güncellenmesi gerektiği için, bu yapının yönetimi ek bir yük getirebilir. Ayrıca, interning yapısının boyutu büyüdükçe, bellek kullanımı ve erişim süreleri artabilir. Sonuç olarak Rust'ta string interning, özellikle büyük ölçekli uygulamalarda bellek kullanımını ve performansı optimize etmek için güçlü bir tekniktir. Bu yazıda, bu tekniğin nasıl uygulanabileceği ve hangi durumlarda ne gibi avantajlar sağlayabileceği detaylı bir şekilde açıklanmıştır. String interning, Rust'un güvenli ve performanslı yapısıyla birleştiğinde, özellikle metin işleme ve sembol yönetimi gibi alanlarda büyük bir potansiyel sunar. Ancak, bu tekniğin uygulanması sırasında dikkat edilmesi gereken noktalar da göz ardı edilmemelidir. Interning uygulamak istendiğinde hazır kütüphanelerin kullanılması veya sıfırdan bir interning

fonksiyonu yazılması programcının ne yapacağına ve hedeflediği şeyin ne olduğuna göre değişebilir. Bu durumda programcı tercihlerine göre bir seçim yaparak interning uygulayabilir.