Harika, istenen metnin transkripti aşağıdadır:

Enes Fehmi usta: Özge Usta hoş geldiniz. Nasılsın?

Özge Usta: İyiyim, teşekkür ederim Enes. Sen nasılsın?

Enes Fehmi usta: Vallahi harikayım. Şimdi birkaç tane soru var, hemen onlara cevap vereyim. Sonra seni tanımakla başlayıp ben arkaya geçeyim, sen bize güzelce SQL'e doğru götüreceksin diye düşünüyorum.

Özge Usta: İle alakalı şeyler var, öyle bir şey olacak.

Enes Fehmi usta: Bu, hangi SQL üzerinden öğretilecek diye soruluyor.

Özge Usta: Hangi SQL? Aslında klasik SQL üzerinden gideceğiz. Pratikte de Google'ın BigQuery'sini kullanacağız. Zaten detaylı eğitim sırasında veriyor olurum.

Enes Fehmi usta: Bir de benim gördüğüm, eğitim ne kadar süreceği ile ilgili bir soru var.

Özge Usta: Ya önceki session 3,5 saat olmuş ama burada o kadar olmayacak, merak etmeyin. Hani en azından böyle 2 saat içinde bitirmeyi planlıyoruz. Bu arada, "Advance kadar gidecek miyiz?" diye sorulmuş. Aslında şöyle, yani evet, advance konulara da götüreceğim ama tabii süremiz kısıtlı olduğu için her konuyu böyle biraz biraz işleyip, birkaç pratik çözüp devam edeceğiz. Bazı başlıkları da tabii ben size bırakacağım sonunda, onları da tabii söylüyor olacağım size yönlendirmek için. O şekilde söyleyebilirim.

Enes Fehmi usta: Bir tane daha, bir şeyle alakalı, data engineer olayı ile alakalı bir soru var. Zaten "İlerleyen zamanda bir bootcamp gelir mi öyle bir şey?"... Ama şu an sadece düşünme aşamasında. Aklımda böyle "full stack data" üzerine bir bootcamp fikri var. Ne kadar olur, nasıl olur, onu ilerleyen zamanlarda göreceğiz diyelim. Daha sonraki soruları sen alırsın deyip, ben arkaya geçeyim, sözü sana bırakayım o zaman.

Özge Usta: Tamamdır.

Enes Fehmi usta: İyi anlatımlar.

Özge Usta: Teşekkürler. Selamlar herkese. Bir ekranı şöyle alayım. Bugün bildiğiniz üzere aslında SQL üzerine konuşacağız, böyle pratikler yapacağız. Aslında böyle sonuna kadar, en azından mesela işte Window Function gibi daha böyle advance konuları da göstermeye çalışacağım.

Önceki oturuma gelmediyseniz kendimi çok kısa bir tanıtayım. Ben 4 seneden fazladır data science alanında çalışıyorum. Şu anda Damaç Teknoloji'de çalışıyorum. O şekilde diyebilirim. O zaman isterseniz hızlıca başlayalım. Zaten zamanımız da, yaklaşık dediğim gibi 2 saat sürmesini planlıyorum eğitimin.

Aslında içerik olarak önce böyle çok kısa bir "SQL nedir?" diye bir konuşacağız. Daha sonrasında temel veri sorgulama üzerine gideceğiz. İşte veri filtreleme, sıralama, koşullu veri dönüşümlerine bakacağız. Sonra önemli konulardan biri olan aslında aggregation function dediğimiz toplama fonksiyonları ve gruplama üzerine bakacağız. Sonraki konumuzda gene önemli konulardan biri, tablo birleştirme yani join yapılarına bakacağız. Daha sonra subquery'lere, yani alt sorguları inceleyeceğiz birlikte. En son bahsettiğim window function üzerine değineceğiz ve ben hani tabii ki süreden dolayı yetiştiremediğim ama sizin de bakabileceğiniz bazı ileri konular ve önerilerde bulunacağım diyebilirim.

Aslında SQL nedir dersek, çok kısa bir şekilde datayı, veriyi oluşturma, okuma, güncelleme ve silmek için kullanılan bir komutlar bütünü diyebiliriz. Burada farklı farklı veritabanı yönetim sistemleri kullanılabiliyor. Bugün biz de zaten onlardan bir tanesi olan Google BigQuery kullanacağız. Onun dışında aslında farklı farklı SQL'ler duyacaksınız. İşte MySQL duyacaksınız, Postgre SQL veya farklı farklı duyabilirsiniz. Bunların evet, ufak sintaks farklılıkları var fakat temelde aslında hepsi birbirine çok benziyor. Yani siz eğer bir tanesini öğrenirseniz diğerlerini yazmanız sizin için çok kolay olacaktır.

Başlamadan önce ben üzerine çalışacağımız bir veri setimiz var, onu ben size çok kısa bir tanıtmak istiyorum. Çünkü bu biraz daha bu arada teori kısmını az tutup daha çok pratik kısma odaklanmak istiyorum. Burada biz Kaggle'da bir "online_shop" diye bir verimiz var. Onun üzerinden... Neden bu veriyi seçtim derseniz de birincisi, zaten bu arada isterseniz siz de bu veriye çok rahat bir şekilde erişebilirsiniz. İkinci olarak da bu verinin güzel yanı, aslında buradaki bütün tabloların birbirine bağlantılı olması. Bu sayede de aslında biz işte mesela Customer tablosu... Şuradan anlatayım. Şimdi ben 5 tane tablo seçtim buradan.

Bu 5 tane tablonun hepsi birbirleriyle belli ID'lerle bağlı. Customer tablomuz için... Customer tablomuz var. Customer tablomuzun primary key'i CustomerID. Primary key nedir derseniz, aslında primary key, her tabloyu, satırlarını unique bir şekilde tanımlayan bir ID'dir. Tabii bazı durumlarda bu primary key iki tane veya daha fazla kolondan da oluşabilir. Mesela Customer tablosunu düşünürsek, yani her müşterinin kendine özgü bir ID'si var. Bunlar da CustomerID olarak tanımlıyorum. Mesela Orders'a gittiğimizde, burada OrderID bizim için unique olan yani primary key. Order_Items'ta OrderItemID. Daha sonrasında da burada Product var ve Supplier. Aslında biz de çalışmalarımız boyunca bu 5 tane veri setini kullanarak devam edeceğiz.

O zaman çok kısa başlayalım. Öncelikle temel veri sorgulama kısmı var. SELECT ve FROM. SELECT bizim için en temel veri sorgulaması. Burada biraz, burada İngilizce kelimelere az çok hakimseniz aslında SQL'i anlamak gerçekten de çok zor olmuyor. Ya bu da SELECT dediğimiz bizim seçeceğimiz kolonları belirliyor. Burada yıldız (*) dediğimizde biz tablodaki bütün kolonları seçmiş oluyoruz. FROM da hangi tablodan, yani SELECT * FROM işte X bir tablo ismi. Mesela dediğim gibi, burada yıldız yerine kolon isimleri yazarsanız da "ben bu bu kolonları seçiyorum" demiş olursunuz. Burada best practice olarak yıldız yazmak aslında çok da doğru olmuyor. Çünkü mesela siz bazen belki 20, 30, 100 tane kolonu olan tablolarda çalışacaksınız ve burada yıldız attığınızda bütün kolonları çekmiş olacaksınız. O yüzden de aslında en doğru kullanımı bu şekilde kolonları ayrı ayrı yazmak. DISTINCT eklediğimizde, burada ekstra olarak buradaki çoklamaları aslında önlemiş oluyoruz. Buradan nasıl bir örnek verebilirim? Mesela bir Orders tablomuz var... Deminki şuradaki hemen yandaki örnekten de açıklayabilirim. Mesela Product tablomuzda, ProductID'ye özgü bir tablo yani her product'ın olduğu, sıralandığı bir tablo. Ama ben burada sadece kategorileri çekmek istiyorum. Eğer burada DISTINCT yazmazsam bütün... DISTINCT yazdığımda aslında sadece o unique olan ikili gruplar gelecek.

Daha sonrasında ikinci en önemli kısımlardan biri de veri filtreleme, yani WHERE kısmı. Burada da adından anlaşılacağı gibi WHERE aslında bizim hani bir condition, bir koşul belirlediğimiz bir kısım. Bu arada ara ara terimlerin hem Türkçelerini hem de İngilizcelerini kullanacağım çünkü bazen yani işe girdiğinizde veya staj yaparken bunların belki çoğu zaman İngilizceleriyle karşı karşıya kalacaksınız. O yüzden de özellikle ikisini de söylemeye çalışacağım. Mantıksal operatörlerimiz aslında AND ve OR oluyor. Aslında yani biraz zaten Python gibi dilleri biliyorsanız bunlara az çok hakimsiniz. AND dediğimizde iki koşulu ya da işte bağladığımız bütün koşulların hepsini sağlaması gerekiyor. OR dediğinizde de bunlardan bir tanesini sağlaması bizim için yeterli oluyor. Bir de farklı karşılaştırmalar var. Temel karşılaştırmamız zaten büyüktür, küçüktür, eşittir, eşit değildir gibi ya da büyük eşit, küçük eşit gibi karşılaştırmalar. BETWEEN dediğimiz yani arada kısmı. Şurada mesela bir örneğimiz var: Price BETWEEN 500 AND 1000 gibi bir aralık verebiliriz. Yani bunu şu şekilde de bu arada yazabiliriz: Price > 500 AND Price < 1000 diye de yazabiliriz ama bu şekilde yazmak tabii çok çok daha, hani hem görüntü anlamında çok daha iyi hem de kafa karışıklığını çok daha azaltıyor. Mesela burada bir IN'le örnek verdim. Burada mesela kategori, kategoriyi ya "Home & Kitchen" olsun dedim ya da "Furniture" olsun dedim. Bu IN'in içinde de bunların içinden birinin olmasını istiyoruz. IS NULL, IS NOT NULL kısmı da bazen kolonların null olup olmadığını check etmemiz gerekiyor. O yüzden bu da bayağı önemli bir karşılaştırma diyebilirim. LIKE biraz daha string dediğimiz yani text verilerinde kullandığımız bir fonksiyon. Yani mesela şuradaki örnekten gidersek "Home & Kitchen" dedik ama mesela bunun tam adını bilmiyorsunuz ama içinde "Home" geçtiğini biliyorsunuz. Mesela Category LIKE, işte tırnak içinde gene yüzde işaretleri var, yani "Home"u onun içine alıp, yani "Home" bunun içinde bulunsun gibi bir şeye bakmak istiyorsanız LIKE da kullanabilirsiniz. LIKE ile ilgili yani biraz daha detaylı araştırma yapmak isterseniz bence bakmanızı tavsiye ederim. Orada o bahsettiğim yüzde dışında farklı şeyler de var.

Daha sonra veri sıralamaya geliyorum. Veri sıralama kısmında ORDER BY bizim için en önemli fonksiyonlardan biri. Burada ASC (ascending) dediğimiz küçükten büyüğe sıralama, DESC (descending) de büyükten küçüğe sıralama gibi düşünebiliriz. Eğer buraya hiçbir şey yazmazsak default'u bunun ASC'dir. Yani şu andaki örneğe bakarsak mesela ORDER BY TotalPrice DESC demesi yani TotalPrice'ı en yüksekten en düşüğe kadar sıralıyoruz burada. Buraya ben hiçbir şey yazmasaydım, yani ASC de DESC de yazmasaydım, bu default olarak ASC yapar, küçükten büyüğe sıralardı. Burada LIMIT 5 ne yapıyor? LIMIT 5 dediğimizde, yani TotalPrice'ı en yükseğe kadar, en yüksekten en düşüğe sırala ve ilk 5'i al diyorum. Peki bu ORDER BY'ı hiç kullanmadan LIMIT atsaydım ne olurdu? Veri kendi sıralanır, oradan hiçbir SELECT FROM yazmışım gibi, sonra en üstteki 5'i alırdı. Burada bu LIMIT'i işte şu şekilde de kullanabiliyoruz. Mesela bir tablo var, yeni bir tablo ve bir tabloyu incelemek istiyorsunuz, ne tarz veriler olduğunu. Ama tablonun normal boyutu belki milyonlarda, ki böyle şeylerle işte tabii ki de çok karşılaşacaksınız. Burada LIMIT 5 atıp, 10 atıp, hani verinin sadece çok az bir kısmını alıp hem maliyeti azaltıp, çünkü siz bütün o milyonluk veriyi çekerseniz, bu tarz o çekme işlemleri çok maliyetli bir işlem. O yüzden de bu LIMIT'i bu tarz böyle "verinin azını çekip bir bakayım" demek için de kullanıyoruz.

Bir sonraki konumuzda koşullu veri dönüşümü. Burada bir IF'ten bir de CASE WHEN'den bahsedeceğim. IF, aslında bir condition yazıyoruz. Eğer bu condition sağlıyorsa value 1'e gidiyor, sağlamıyorsa value 2'ye. Aşağıdaki örnekte de açıklayabiliriz bunu. Ne demişiz? SELECT demişiz ve OrderID'miz var. "Eğer" diyorum... Bu arada şurada bir virgül olacak, siz yapmadan ben söyleyeyim, orada eksik kalmış bir virgül. Burada bütün kolonları tabii de virgülle ayırmamız gerekiyor. Oradan hani bunu bir şekilde çalıştırmaya çalışırsınız, mesela size bir böyle hata verip hata gösterecektir. IF'in içine ben condition'ı yazıyorum. Diyorum ki TotalPrice diyorum 500'den küçükse, diyorum, böyle bir isim ver: Low. Bu arada bu isim değil de bir, yani bir sayı da olabilir. Buraya direkt ben düz 500 de yazabilirdim. Eğer buysa bunu yaz, bu değilse, bunu sağlamıyorsa High. Buraya da PriceSegment diye isimlendirdim. Bu arada buraya AS yazabiliriz. AS yazdığımızda aslında bir isim atamış oluruz ama AS yazmadan bu şekilde de yazabiliyoruz ama best practice olarak AS yazmak her zaman çok çok daha tabii iyi oluyor.

CASE WHEN şunu sağlıyor. Aslında IF'in daha gelişmiş versiyonu gibi biraz düşünebiliriz. Çünkü burada farklı farklı bir sürü condition ekleyebiliyorsunuz. İsterseniz buraya 20 tane condition ekleyebilirsiniz. ... CASE WHEN'de dediğim gibi, burada aslında birden fazla condition yazmamızı sağlıyor. Bu CASE WHEN'le ilgili bir örnek çözeceğim zaten daha iyi kafanıza otursun diye. Benim de sık kullandığım fonksiyonlardan biri. Ya tek bir condition olduğunda ben IF'i kullanmayı tercih ediyorum çünkü çok basit bir yazımı var. Fakat birden fazla condition olduğunda IF'in içine böyle IF'in içine IF koyuyorsunuz ve orası çok karmaşık bir hale geliyor. O yüzden de birden fazla condition'ınız varsa buradan IF'ten CASE WHEN'e dönmeniz çok daha doğru olur.

Devam edelim. Şimdi asıl ben buradan önce bir pratik koyacaktım ama bu aggregation function olmadan da örnek yapmak da çok mantıklı olmuyor açıkçası. Çünkü gerçekten çok temel veri... Çok temel yapı taşlarından biri. Söyleyebileceğimiz aggregation function ne? Aslında gerçekten bir toplama yapıyoruz burada. Yani bir önemli bir GROUP BY kısmı var. Yani burada belirlediğimiz belli kolon isimleri var. Bu 1 ya da birkaç tane kolon olabilir. Bu kolonlara göre, bu kolonları gruplayarak burada bir toplama fonksiyonu kullanıyoruz. Yani bu toplama fonksiyonu ne olabilir? Mesela SUM olabilir yani o değerleri toplama. COUNT olabilir yani o değerleri saymamız gerekebilir. MIN'ini alabiliriz, MAX'ını alabiliriz, AVG (average) alabiliriz, standart deviation'ını alabiliriz ya da burada yapılabilecek orada farklı farklı fonksiyonlar var. Mesela örnekte de şöyle yapmışız. Customer ve Orders tablomuz var. Orders tablomda ben her Customer için TotalPrice'a bakmak istemişim, orada da onu SUM'lamışım. Sonra GROUP BY diyerek... Burada bu GROUP BY 1 dememin sebebi şu, burada her, işte bu birinci indeks, bu ikinci indeks gibi düşünürseniz, bu 1 aslında CustomerID'yi temsil ediyor bizim için. Buraya isterseniz 1 yerine CustomerID de yazabilirsiniz. O da kesinlikle doğru bir kullanım. Mesela şunu sorarsanız, "bu ikisinin yeri değişirse ne olacak?" derseniz, CustomerID ikinci sıraya geldiğinde bu GROUP BY 2 olur. Buradaki kolon sayısı arttıkça tabii ki de buraya ekleme yapıp yani burada tüm kolonları ya numarasını, yani indeksini ya da kolonun kendisini yazmanız gerekiyor GROUP BY dediğimizde.

Evet, şimdi birkaç soru üzerine geçeceğiz. ... Okey, o zaman bir veriyi tanımaya başlayalım. Yani şöyle, bu arada elinizde mesela herhangi bir case geldi diyelim veya bu arada bu live coding de olabilir veya mesela bir şirket size bir case attı, hani nasıl ilerlemeli SQL yazarken, biraz ona da değinmek istiyorum. Bizim elimizde dediğim gibi 5 tane tablomuz var. Önce bu 5 tane tabloyu birlikte hızlıca bir inceleyelim başlamadan. ... O zaman devam ediyoruz. Burada Customer tablomuz. CustomerID bizim primary key'imiz. Customer için işte bazı demografik bilgilerimiz var ve işte isimleri, adresleri, e-mailleri, telefon numaraları gibi. Orders tablosuna baktığımızda burada bizim siparişimiz hakkındaki detaylar var. İşte siparişin ID'si, ne zaman sipariş verilmiş, hangi müşteri tarafından ve total price'ı. Order_Items dediğimiz, burada Orders ile farkı şu, Orders tablosuyla... Şimdi siz bir siparişin içine birden fazla ürün aldığınızda, her bir ürün için, o siparişte alınan ürün için ayrı bir satır oluşturuluyor ve burada o ürün hakkında bilgi veriliyor. Yani burada OrderItemID bizim primary key'imiz oluyor. Hangi order olduğu, işte bu ProductID nedir, bu product'tan kaç tane sipariş edilmiş ve bu product'ın purchase sırasındaki price'ını alıyoruz. Daha sonra Product tarafında da ProductID bizim primary key'imiz, ismi, kategorisi, fiyatı. Daha sonrasında Supplier tablomuz var. Supplier'da da SupplierID'miz var, supplier'la ilgili diğer detay bilgilerimiz var.

Okey, o zaman birlikte biraz yazmaya başlayalım. ... Önceki şeyi bulalım istiyorum, her kategori için kaç farklı ürünümüz var, kaç farklı supplier'ımız var ve kategori başına ortalama ürün fiyatlarına bakalım. Şimdi bizim elimizde product'ın içinde kategori bilgimiz var. Bununla ilgili verilerimizi çekelim. Mesela SELECT... Ben her zaman ilk bir base'imi oluştururum. FROM Product. ... SELECT Category... Şimdi kaç tane kategoriye ait ürün olduğuna bakalım. COUNT(DISTINCT ProductID) dedim. Bu arada burada aslında her ProductID unique olduğu için buraya ben DISTINCT yazmak zorunda da değilim. Neden yazdığımı da açıklayacağım aslında şimdi. Burada aynı sayı denk geliyor ama şunun farkını da birlikte konuşalım istiyorum: COUNT(*), COUNT(ProductID), COUNT(DISTINCT ProductID). Bu üçünün farkı şu oluyor: DISTINCT dediğinizde unique olan sayıyı topluyor size. ... COUNT(ProductID) dediğimizde unique'liğe bakmadan direkt sayıyı topluyor. Peki yıldızla bunun ne farkı var derseniz, bu şekilde yazdığınız zaman null olan ProductID'leri saymaz ama yıldız yazdığınızda null olup olmamasının bir önemi

yoktur ve hepsini sayar. ... Kaç farklı supplier var dedim. Farklı supplier dediğim için burada gene DISTINCT almam gerekiyor. ... Sonrasında da kategori başına ortalama ürün fiyatına bakacağız. Burada da average (AVG) alıyoruz. ... En sonunda bu arada GROUP BY Category her türlü zaten yazmak zorundayız. ... Çalıştırdığımızda işin sonunda ne olacak? Dört tane kategorimiz var, product count'lar bu kadarmış, işte supplier'larımız... Mesela ben bunu DISTINCT yazmasaydım buraya, şu sayı burada da çıkacaktı. O yüzden burada DISTINCT yazmak gerçekten önemli.

Bu sefer bir de şeyi göstermek istiyorum. CASE WHEN dediğim gibi önemli. O yüzden burada şuna bakalım istiyorum: Siparişleri toplam tutarlarına göre bir gruplayalım ve her grup için bir sipariş adedi bulalım. Aslında hem CASE WHEN'i kullanacağız hem de GROUP BY'ı kullanacağız birlikte. ... Önce bir tablomu da alayım ben, burada Orders tablosunu kullanacağım. ... CASE WHEN TotalPrice < 1000 THEN '0-1000'... Şurada 1000'den büyük olduğunu yazmadık. Çünkü bunun sebebi aslında CASE WHEN'de SQL şunu anlıyor: "Zaten sen diyor, 1000'den küçük olanları burada yazdın, geri kalan, hani 2000'den küçük olanların zaten 1000'den büyük olması lazım." O yüzden buraya benim tekrardan "hani 1000'den büyükse" gibi bir şey yazmama gerek yok. ... Geri kalan, hani ben 3000'in üstüne de ELSE gibi bir şey yazmak istiyorum. Mesela burada gerçekten ELSE'i kullanıyoruz. "Bunların üçü de değilse" diyorum, "o zaman diyorum, bunun adı işte 3000 Üstü olsun" diyorum. ... AND ile bunu bitiriyoruz. Buraya gene bir isim... Pardon END. Burada END yazmamız gerekiyor. ... END AS PriceSegment diyelim bunun adına. Ne yaptık? Her segmentin içine ne kadar sipariş olduğunu bulmak istiyorduk. Orada da gene bir COUNT yazalım. Burada OrderID'yi toplayalım. ... GROUP BY 1 diyeceğiz. Çünkü artık şurası bizim için yeni bir kolon oldu. Mesela bunu çalıştırdığımızda şu şekilde görebiliriz.

Biraz orayı atlamış gibi oldum, şuradaki HAVING kısmını. Orayı da anlatayım. ... HAVING nedir? Şimdi WHERE condition'ını gördük. WHERE aslında bir condition, bir filtreleme belirtiyor bizim için. HAVING de belirtiyor ama HAVING'i sadece GROUP BY'da kullanabiliyoruz. Yani mesela şöyle bir örnek yapalım birlikte. Sorumuz mesela neymiş? 10'dan fazla ürün içeren tüm siparişlerin ID'lerini alalım. Gene Order_Items tablosuna gidelim birlikte. ... SUM(Quantity) AS TotalProduct. ... WHERE'de normalde FROM'un altında, GROUP BY'ın üstündedir. WHERE TotalProduct > 10 dersem mesela, bir hata alacağım. ... WHERE'de bizim aggregate fonksiyonları filtrelemesi yasak. O yüzden de WHERE kabul etmeyecek. Bunu alacağız, bunu HAVING'e koyacağız. Aslında dediğim gibi, WHERE'le HAVING'in mantığı böyle tamamen filtreleme anlamında aynı. Ama fark olarak, HAVING sadece aggregate fonksiyonlarda kullanılıyor. ... Aynen, mesela şu an sadece 10'un üstünde total product olan siparişleri seçmiş olduk.

O zaman JOIN kısmıyla devam edelim. JOIN gerçekten çok kritik bir fonksiyon bu arada. Ve özellikle burada data analisti ya da scientist taraflarına da işe girmek isteyen arkadaşlar, kesinlikle JOIN mantığını çok iyi öğrenmeleri gerekiyor. Burada dört temel JOIN'den bahsedeceğim. INNER JOIN dediğim, aslında şöyle, JOIN şu oluyor: Yani bir sizin Tablo 1'iniz var, Tablo 2'niz var. Bu iki tabloyu ortak olan kolonlar sayesinde birleştiriyorsunuz. Ama bunu yaparken de farklı şekillerde bu JOIN işlemini yapabiliyorsunuz. Yani INNER yaptığınızda Tablo 1 ve Tablo 2'nin tamamen ortak kesişim kümesini alıyorsunuz. LEFT JOIN dediğinizde Tablo 1'i tamamen, RIGHT JOIN'de Tablo 2'yi eklerken sadece Tablo 1'i alıyorsunuz ve ikisinin ortak kesişimi oluyor. RIGHT dediğimiz aslında LEFT'in tamamen tam tersi ama genelde best practice olarak hep LEFT JOIN'i kullanırız. Çünkü Tablo 1 her zaman ana, sizin temel tablonuzdur. Tablo 2 onun yanına eklendiği için genelde RIGHT çok tercih edilmez. ... FULL OUTER JOIN, diğer adıyla aslında FULL JOIN olarak da kullanabilirsiniz. Bu da ikisinin birleşimi, yani iki tablodaki de bütün verileri alan yapı diyebiliriz. Burada ana yapımız şu şekilde: SELECT * FROM Table1. Dediğim gibi, ana tablomuz... Sonra INNER JOIN Table2 ON Table1.ID = Table2.ID...

Şimdi soruları hem siz de görmüş olursunuz. ... Şimdi, az önce kategorilerle ilgili ufak bir hesaplama yaptık birlikte. Şimdi bunu bir tık üste taşıyalım ve her kategori için toplam geliri hesaplayalım, satılan ürün sayısını hesaplayalım, bir de kaç farklı kullanıcının bu kategoriden alışveriş yaptığını hesaplayalım. Şimdi bunu yapmak için aslında bizim birkaç tabloyu birleştirmemiz gerekiyor. ... Hepsini aslında şu an burada INNER JOIN ile birleştirebiliriz. ... Benim buradaki ana amacım neydi? Kategori üzerinden farklı metrikasyonlar yapmak. ... Burada neler bakıyorduk? Toplam gelire bakacağız. Toplam gelirimiz aslında neydi? Orders tablosundan TotalPrice'ı alabiliriz. ... Satılan ürün sayısına bakıyoruz. Burada Quantity'den gidiyoruz. ... Sonraki hesabımız neymiş, hemen ona da bakalım. Kaç farklı kullanıcı? Burada "kaç farklı" dediğimiz için burada gene COUNT(DISTINCT CustomerID) diyorum. ... Burada tabii ki ben gruplamayı henüz yapmadığım için hemen bana bir hata verdi. Biz de tamam deyip hemen bir GROUP BY Category ekliyoruz sonuna.

Bakalım bir sonraki sorumuz neymiş. "Hiç satışı olmamış ürünlerin isim ve kategori bilgilerini alalım." Şimdi, "hiç satışı olmamış ürünler"... Bu da biraz iş hayatında kullanılan bir yapı bu arada. ... Şimdi, satış bizim için neydi? Ürün satışına baktığımız Order_Items'tı. ... Şimdi Product'ı base olarak alıyorum. ... Bunu LEFT JOIN olarak bağlıyorum, Product'ı kaybetmiyorum. Onu onun yanına ekliyorum gibi düşünün. Bunu da ne üzerine ekliyorum? ProductID üzerine. ... Hangi

bilgileri istiyordu bizden? İsim ve kategori bilgisini. ... Burada şunu yapacağım: WHERE oi.ProductID IS NULL diyorum. Şimdi bunu neden böyle dedim derseniz, şöyle oluyor. Şimdi dedim ki burada 30 tane product var, ben bu 28 tane olan kısmı birleştirdim burayla. Bu Tablo 1, bu Tablo 2'yi buna ekledim. Burada bazıları eksik olduğu için maalesef Tablo 2'deki ProductID'ler null gelecek. Ben de şunu söylüyorum: Bunun null geldiği durum, aslında Tablo 2'de olmayıp Tablo 1'de olan kısımlar olacak. ... Otomatik olarak Product tablosunda olan ama herhangi bir siparişi olmayan ürünleri almış oluyorum.

Bir sonraki soruya gelelim. "İçinde 'city' geçen adreslerin her birine kaç farklı tedarikçi ürün satmış, bulalım" diyoruz. Şimdi, adres dediğim zaten Customer adresi. ... Kaç supplier dediğimiz için... Yani kaç farklı supplier. ... Bizim tedarikçiye ulaşmamız için, tedarikçiyi gördüğümüz aslında bizim ilk kolon, Supplier dışında bir de Product tablomuz var. Product tablosuna kadar aslında gitmemiz gerekiyor. ... Orders'a gitmemiz lazım. Bunun sebebi ne? Orders'ta CustomerID var. ... Ürüne gitmemiz lazım ki supplier'a gidebilelim. ... En son olarak da Product'a gidiyorum. ... Product'la da burada ProductID ile bağlıyorum. Şimdi ne demiştik? Adresler için kaç farklı supplier'a gidiyoruz. ... COUNT(DISTINCT SupplierID) atıyorum gene. ... Burada tabii ki de GROUP BY Address unutmuyoruz. ... Bir de ben şunu eklemiştim ekstra olarak: "içinde 'city' geçen adresler". Burada ufak bir LIKE göstermek istediğim için bunu ekledim aslında. WHERE Address LIKE '%city%'. ... Böyle yaptığınızda aslında büyük küçük... Yani benim elimde hepsi büyük olduğu için ben direkt düz bir şekilde istersem az önceki gibi de yazabilirim. Ama emin değilseniz... LOWER(Address) LIKE '%city%'. ... Çalıştırdığımızda aynen, sadece elimizde gördüğünüz gibi "city" olanları bize dönmüş oldu.

O zaman buradaki kısım da bitti. Şimdi gene böyle farklı, önemli konulardan biri olan alt sorgulara dönüyorum. Aslında bu alt sorgular neden önemli? Evet, bazı durumlarda işte tabloları birbirine bağlamak için hani JOIN'leri kullanıyoruz ama bazen JOIN'ler yetersiz kalabiliyor, çok daha karmaşık problemlerimiz olabiliyor. ... O yaratacağınız tablo sadece ara geçiş bir tablo oluyor. Siz de boşuna bu ara geçiş tabloyu yaratmamak için aslında query yazarken içine böyle subquery'ler ekliyoruz. ... Diğeri WITH yapıları. Aslında bu CTE dediğimiz Common Table Expressions oluyor. Burada WITH Base AS (...) deyip kendiniz bir isim takıyorsunuz bu arada. ... Bir tane query'mi yazıyorum. ... Bunu daha sonra kullanabiliyorsunuz ya da bunu alıp işte başka bir tabloya JOIN'leyebiliyorsunuz.

"Hiç satışı olmamış ürünleri tedarik eden tedarikçilerin bilgilerini getirelim." Şimdi biz zaten hiç satışı olmamış ürünleri bulduk değil mi, az önce bir tane query'mizi yazmıştık. Şunu alalım hatta direkt. ... WITH UnsoldProducts AS (...). Bu query'yi koydum. ... Ne yapıyorum? SELECT * FROM Supplier s JOIN UnsoldProducts up ON s.SupplierID = up.SupplierID. ... Sadece o 3 ürüne ait 3 tane supplier varmış, üçü de farklıymış.

Bir sonraki örneğe bakalım. "Öncelikle, sattığı toplam ürün miktarına göre en çok ürün satan 3 tedarikçiyi bulalım" diyor. "Sonra bu tedarikçilerin hangi ürünlerden ne kadar sattığını bulalım." Aslında burada iki tane task var fark ettiyseniz. Önce bir 3 tane tedarikçi bulmam gerekiyor benim, en çok ürün satan. Daha sonra diyor ki, "bu tedarikçilerin hangi ürünlerden ne kadar sattığını bulalım" diyor. O yüzden de burada WITH yapısı kullanmamız gerekiyor. Yani burada WHERE kullanmak, JOIN kullanarak pek çözebileceğimiz bir işlem değil bu. O yüzden mesela WITH TopSuppliers AS (...). Benim buradaki ilk query'min amacı top supplier'ları bulmak. ... ORDER BY TotalProductCount DESC ... LIMIT 3 diyerek aslında en çok ürün satan üçünü buldum. ... Daha sonrasında ne yapıyoruz dedik? Bu tedarikçilerin hangi ürünlerden ne kadar sattığını bulalım diyor. ... Ekstra olarak benim geri kalan supplier'lara ihtiyacım olmadığı için şuraya INNER JOIN TopSuppliers atıyorum ki hani bu sayede zaten INNER JOIN aldığımda geri kalan bütün supplier'lar elimine oluyor.

Buradaki son sorumuza geçelim. "Her müşterinin 1. siparişleri dışındaki tüm siparişlerinin total hacmini bulalım." Şimdi, birinci sipariş dediğimizde ne oluyor? Şimdi bunu farklı şekillerde bulabilirsiniz. Ben sadece bir yöntemi göstereceğim. ... "Herkesin bir, birinci siparişini bulalım, sonra da bunları diğer siparişlerden çıkarıp toplayalım." ... WITH FirstOrders AS (SELECT CustomerID, MIN(OrderDate) AS FirstOrderDate FROM Orders GROUP BY CustomerID). ... Orders'a LEFT JOIN atıyorum FirstOrders. ... WHERE fo.CustomerID IS NULL. Aslında bunu dediğimde gene aynı durum oluşuyor. Yani benim ikinci taraftan gelenleri ben elimine etmiş oluyorum.

O zaman window fonksiyonlarından devam edelim. Burası burada bir tık daha artık advance bir seviyeye giriyoruz. Önemli temel fonksiyonları öğrendik. ... Bence mutlaka window fonksiyonlara çalışın. Biraz başta kafa karıştırıcı olabiliyor window function'lar, fakat yazdıkça gerçekten öğreniyorsunuz. ... Ne olduğunu anlatayım. Şimdi GROUP BY'ı biliyorsunuz zaten, artık öğrendik. Belli kolonlar üzerinden GROUP BY yapıyoruz ama kolonlar için... Ama bazen tablonun yapısını değiştirmek istemiyoruz ama gene de o gruplama ve o hesaplamaya ihtiyaç duyuyoruz. Böyle durumlarda bu window fonksiyonlar, yani pencere fonksiyonları dediğimiz fonksiyonlar devreye giriyor. Bunların yaptığı şey aslında yapınızı değiştirmeden, mesela işte gene siz Customer için TotalPrice almak, total getirdiği geliri almak istiyorsunuz ama Customer'da OrderID de dursun

istiyorsunuz. ... Yani böyle durumlarda sizin yapınızı bozmadan o hesaplamaları yapıyor. Ha bir de sıralama kısmı tabii çok önemli. ... Bazen gruplara göre rank'lemeniz gerekiyor yaptığınız işi.

Şimdi az önce hatırlarsanız bir tane örnek çözdük TopSuppliers diye. ... Peki ya şöyle deseydim ben: "Burada sadece top 5, yani her supplier için top 5 product'ı getirelim" gibi bir şey mesela yapmak istedim. Şimdi burada LIMIT kullanamazsınız çünkü öyle olursa bütün hepsini limit alacak. Ben supplier özelinde, hani top 5 product istiyorum. Öyle olunca işler açıkçası biraz karışıyor. O yüzden burada ekstradan bir ROW_NUMBER işin içine giriyor. ROW_NUMBER'ı nasıl kullanıyoruz? ROW_NUMBER aslında çok basit bir şekilde sıra atıyor ve bu sıraya da biz karar veriyoruz. Nasıl karar veriyoruz derseniz, bu PARTITION BY kısmı aslında GROUP BY gibi. Yani neye göre bu ROW_NUMBER'ı atacağım? Aslında ben supplier'a göre atmak istiyorum. ... ROW_NUMBER() OVER (PARTITION BY SupplierID ORDER BY TotalQuantity DESC) AS Rank. ... Çalıştıracağım, ne çıktığına bakalım, öyle daha çok aklınızda bence oluşacak. Mesela SupplierID 537. Ne yapmış? Sırayla bakın, büyükten küçüğe doğru 1, 2, 3 vererek sıralamaya başlamış hepsini. ... Daha sonra "ilk 5'i seçmek istiyorum" dedim. ... Google BigQuery'nin kendisine şöyle bir özelliği var: QUALIFY. Bu da WHERE ve HAVING gibi bu arada ama sadece window function'a özel. ... QUALIFY Rank <= 5 dedim. ... Ama QUALIFY maalesef birçok SQL veri yönetim tabanında yok. O yüzden de bunu yoksa ne yapacağız? Çok basit. Hemen şurada bunun adına Base koyayım... WITH Base AS (...) SELECT * FROM Base WHERE Rank <= 3.

Son örneğimiz, bir tık daha böyle bir karmaşık bir örnek çözeceğim bununla ilgili. ... "Her müşteri için total sipariş gelirini alacağız. Bulunduğu adresin toplam sipariş gelirini alacağız. Bir de zaman içinde kümülatif olarak artan toplam sipariş gelirini tek bir tabloda toplayacağız." ... SUM(TotalPrice) OVER (PARTITION BY CustomerID) ... SUM(TotalPrice) OVER (PARTITION BY Address) ... Kümülatif hesaplamayı da şu şekilde yapıyorum: SUM(TotalPrice) OVER (PARTITION BY CustomerID ORDER BY OrderDate). ... Burada ORDER BY verirseniz artık bu kümülatif bir toplamaya dönüyor.

Pratikler buraya kadardı. Ben biraz ileri konulardan çok kısa bir de size bahsetmek istiyorum. Burada ileri konularda veri manipülasyonu var. ... INSERT, UPDATE, DELETE gibi. ... Onun dışında bence kesinlikle bakmanız gereken bir kısım tarih ve saat fonksiyonları. İşte CURRENT_TIMESTAMP alabilirsiniz ya da EXTRACT dediğimiz... String'ler var, CONCAT dediğimiz işte iki string'i birleştirme gibi. ... Bir de set işlemleri yani UNION, INTERSECT, EXCEPT gibi fonksiyonlar.

Pekiştirme yolları olarak neler yapabilirsiniz? İnteraktif eğitim sitelerini kesinlikle değerlendirin. ... Biraz mülakat odaklı, gerçek hayata yakın zorlukta SQL problemleri çözmek. ... Gerçek dünya verisiyle çalışmak tabii çok daha iyi oluyor. ... Bir de ileri SQL fonksiyonlarını araştırın.

Son olarak birkaç kaynak önerisi vereceğim. Bunların başı W3Schools. Benim 4 sene önce SQL öğrendiğim yerlerden biridir. ... HackerRank, LeetCode gibi yerlerde bu bahsettiğim hani mülakat odaklı, daha böyle gerçek hayat zorlukları güzel oluyor bence. ... Datacamp, benim gene kullandığım bir tool. ... Kaggle'ın da bu arada kendi Advanced SQL eğitimi var, o da ücretsiz, ona da bakabilirsiniz. ... Benden bu kadar, çok teşekkür ediyorum. Enes'i çağırayım.

Enes Fehmi usta: Ağzına sağlık, gerçekten teşekkür ederim. ... Gerçekten bu SQL sorularının derlenişi, yazı şekli falan hoştu. Gerçekten izleyen kişilere de ben çokça katkı sağladığını düşünüyorum o tarafta. Hatta tekrar tekrar dönüp izleyebilirler.

Özge Usta: Ya şöyle, hani bir zaman baskısı da şu, yani ben tabii query'leri önceden hazırlayıp da yazdım ettim ama hani biraz bakış açımı, hani niye o query'yi nerede nasıl yazdığımı da görmelerini gerçekten istedim insanların. Çünkü bu önemli oluyor. Direkt query'ye bakıp "ha tamam bu böyleymiş" demek kolay. ... O yüzden de o yazma sürecini biraz göstermek istedim.

Enes Fehmi usta: Kesinlikle. Genel olarak şey sorayım ben de sana. Mesela şimdi LLM'den destek almaktan bahsettik. Şimdi LLM'ler gerçekten SQL'i çok iyi yazıyor özellikle kolonları, problemi ve işte oradaki şemayı verdiğin zaman. ... Gerçekten LLM'in yazamadığı böyle karmaşık query'lerle çalışıyor musun?

Özge Usta: Var tabii, olmaz mı? Yani ya doğru yazamıyor, hani olabiliyor. Bu arada basit, bu böyle ilk gösterdiğim örnekleri falan çok rahat yazar zaten. ... Ama böyle yazamadığı durumlar ya da yazsa da çok karmaşık yazdığı durumlar oldu. O yüzden de hani aslında Python'da dediğimiz duruma geliyor gene. Sizin bir SQL temeliniz olmalı ki yazdığı kodu değerlendirip "ya bu query çok optimize değil" ya da "bu query doğru sonucu vermiyor" kısmını kendiniz çıkarabiliyor olmanız lazım. ... Çünkü dedim ya, maliyetler yüksek. Hani Google sonuçta her bir query için aslında bir sizden bir para kesiyor.

... (Q&A devam eder) ...

Enes Fehmi usta: O zaman ağzına sağlık diyelim tekrardan. Bütün katılımcılarımıza da hem buraya kadar geldikleri için hem de bizi soruları ile yalnız bırakmadıkları için teşekkür edelim. Herkese iyi akşamlar dileyelim.

Özge Usta: Görüşmek üzere o zaman.

Enes Fehmi usta: Görüşürüz.