**İSMAİL EMRE GÜNGÖR**

**BİLGE ADAM & ZİRAAT TEKNOLOJİ - ORACLE & PLSQL SINAVI**

* **Soru 1:**
* **Normalizasyon Nedir?**
* Normalizasyon veritabanındaki verilerin tutarlı, anlaşılır ve verimli bir şekilde saklanmasını sağlamak amacıyla kullanılan veri tabanı tasarım yöntemidir.
* **Normalizasyonun Amacı?**
* Veri tutarlılığını sağlamak.
* Veritabanı güncelleme anomalilerini önlemek.
* Veri yapısını esnek ve ölçeklenebilir hale getirmek.
* **Normalizasyona Neden İhtiyaç Duyulur?**
* Veri İntegritesi: Veri tekrarının önlenmesi ile verilerin çelişkisiz ve güvenilir olması sağlanır.
* Anomalilerin Önlenmesi: Güncelleme, ekleme veya silme işlemlerinde oluşabilecek hataların önüne geçilir.
* Disk Alanından Tasarruf: Redundansın azalması disk alanı kullanımını verimli hale getirir.
* Performans Artışı: Sorgulama ve diğer veritabanı işlemleri daha hızlı ve verimli olabilir.
* Esneklik ve Ölçeklenebilirlik: Veritabanının zamanla büyümesine ve değişen ihtiyaçlara uyum sağlamasına olanak tanır.
* **Normalizasyon Çeşitleri (Normal Formlar)?**
* Birinci Normal Form (1NF)
* İkinci Normal Form (2NF)
* Üçüncü Normal Form (3NF)
* Boyce-Codd Normal Form (BCNF)
* Dördüncü Normal Form (4NF)
* Beşinci Normal Form (5NF)
* **Soru 2:**
* **Foreign Key Nedir?**
* Foreign Key (Yabancı Anahtar), bir veritabanı tablosunda, başka bir tablonun primary key'ine referans veren bir alandır.
* **Foreign Key Kullanma Amaçları?**
* İlişkisel Bütünlük Sağlamak: Foreign Key, bir tablodaki sütun veya sütunlar, diğer bir tablonun anahtar sütunlarına (genellikle birincil anahtar) referans verir. Bu referans mekanizması sayesinde, ilişkisel veritabanlarında tablolar arası ilişkiler kurulur ve veri bütünlüğü sağlanır.
* Veri Tutarlılığını Koruma: Referans verilen değerlerin var olmasını zorunlu kılarak, tablolar arasındaki tutarlılığı garanti eder. Örneğin, bir sipariş tablosundaki müşteri ID'si, müşteriler tablosundaki bir kayıda karşılık gelmelidir.
* Redundansı Azaltma: Foreign Key sayesinde, tekrarlanan verilerin saklanmasını önleyerek veri tekrarını ve gereksiz yer kullanımını azaltır.
* Güncelleme ve Silme İşlemlerinin Yönetimi: Bir kaydın referans verilen tablodan silinmesini veya değiştirilmesini denetler.
* **Örnek Senaryo?**

CREATE TABLE Musteriler (

MusteriID int PRIMARY KEY,

MusteriAdi varchar(255),

MusteriSoyadi varchar(255)

);

CREATE TABLE Siparisler (

SiparisID int PRIMARY KEY,

MusteriID int,

SiparisTarihi date,

FOREIGN KEY (MusteriID) REFERENCES Musteriler(MusteriID)

);

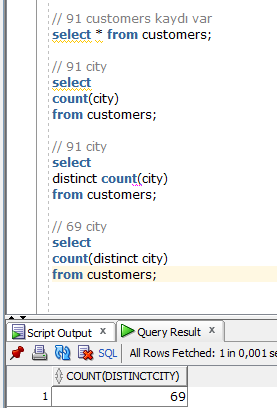
* Bu örnekte, Siparisler tablosunun MusteriID sütunu Musteriler tablosunun MusteriID sütununa Foreign Key olarak bağlanmıştır.
* Bir Sipariş kaydı eklenmek istendiğinde, eğer MusteriID Musteriler tablosunda mevcut değilse, sistem hata verecektir.
* Bu sayede, her siparişin geçerli bir müşteriye ait olduğundan emin olunur.
* **Soru 3:**
* Select Distinct Count(SutunAdi)

From TabloAdi;

* Bu sorgu, öncelikle SutunAdi için toplam kayıt sayısını hesaplar. Bu kullanımda, DISTINCT'in etkisi yoktur.
* Select Count(Distinct SutunAdi)

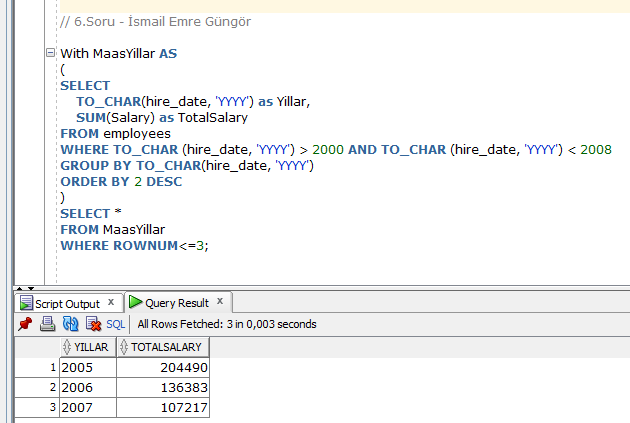
From TabloAdi;

* Bu sorgu, TabloAdi'ndeki SutunAdi sütunundaki tekrar etmeyen (benzersiz) değerlerin sayısını döner. Önce DISTINCT anahtar kelimesi, SutunAdi içindeki tekrar eden değerleri filtreler ve yalnızca benzersiz olanları tutar. Ardından, COUNT() fonksiyonu bu benzersiz değerlerin sayısını hesaplar.
* **Örnek:**



* **Soru 4:**
* **Primary Key Tanımlaması:**
* Bu ifade, isimid sütununu AliT\_isimler tablosu için birincil anahtar (Primary Key) olarak tanımlar. primary key yani birincil anahtar, tablodaki her kaydı benzersiz ve tanımlanabilir hale getiren özel bir kısıtlamadır (constraint).
* Constraint pk\_isimidA1 Primary Key(isimid) satırı AliT\_isimler tablosunda isimid sütununu eşsiz ve boş olamaz olarak tanımlar ve bu sütunun tablo içinde benzersiz kimlik sağlamasını garanti eder. Bu yapı, tablonun veri bütünlüğünü korur ve ilişkisel veritabanı tasarımının temel taşlarından biridir.

* **Soru 5:**
* **SavePoint nedir ve ne işe yarar?**
* Oracle'da SAVEPOINT, bir işlem içinde belirli bir noktayı işaretlemek için kullanılır. Bu işaretlemeler, bir işlemin belirli bir noktasına geri dönülmesine olanak tanır.
* **Savepoint Kullanma Amaçları?**
* İşlem Noktası Tanımlama: SAVEPOINT A; ifadesi, PersonellerA tablosuna bir ekleme yapıldıktan sonra bir işlem noktası oluşturur. Bu noktadan sonra yapılan işlemler, istenirse bu noktaya kadar geri alınabilir.
* Birden Fazla İşlem Noktası Tanımlama: Yukarıdaki örnekte, A, B, ve C olarak üç farklı SAVEPOINT tanımlanmıştır. Her bir insert işleminden sonra ayrı bir SAVEPOINT oluşturulur, böylece her insert işlemi için geri dönüş noktası belirlenmiş olur.
* Hata Yönetimi ve Kısmi Geri Alma: Eğer insert işlemlerinden birinde bir hata oluşursa veya verilerde beklenmedik bir sorun oluşursa, ROLLBACK TO SAVEPOINT A; komutu ile A işaretine kadar olan tüm işlemler geri alınır, ancak A SAVEPOINT'inden önceki işlemler etkilenmez.
* İşlem Esnekliği: SAVEPOINT komutları, uzun ve karmaşık işlemlerde esneklik sağlar. Bir hata meydana geldiğinde veya belirli bir işlem adımından memnun kalmadığınızda, tüm işlemi baştan yapmak yerine belirli bir SAVEPOINT'e geri dönebilirsiniz.
* Güvenli Nokta Oluşturma: SAVEPOINT'ler, bir işlem sırasında "güvenli noktalar" oluşturarak, işlemin bu noktalara kadar olan kısmının kalıcı hale gelmesi için bir COMMIT işlemi yapılana kadar bekletilmesini sağlar.
* **Soru 6:**



**With MaasYillar AS**

**(**

**SELECT**

**TO\_CHAR(hire\_date, 'YYYY') as Yillar,**

**SUM(Salary) as TotalSalary**

**FROM employees**

**WHERE TO\_CHAR (hire\_date, 'YYYY') > 2000 AND TO\_CHAR (hire\_date, 'YYYY') < 2008**

**GROUP BY TO\_CHAR(hire\_date, 'YYYY')**

**ORDER BY 2 DESC**

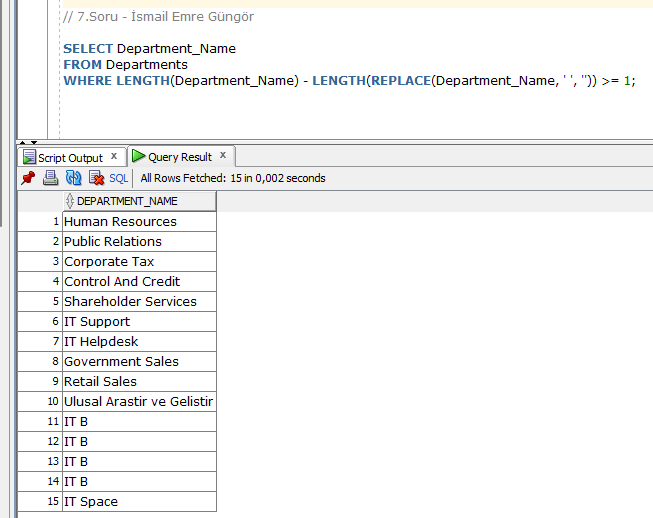
**)**

**SELECT \***

**FROM MaasYillar**

**WHERE ROWNUM<=3;**

* **Soru 7:**

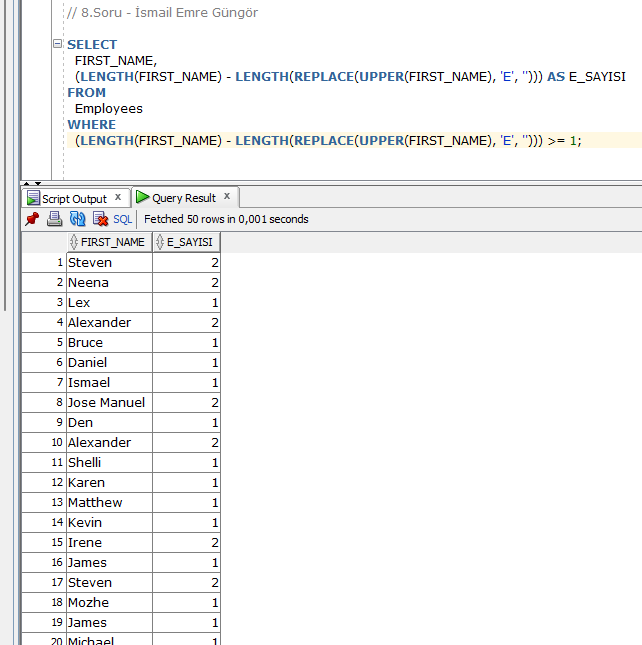


**SELECT Department\_Name**

**FROM Departments**

**WHERE LENGTH(Department\_Name) - LENGTH(REPLACE(Department\_Name, ' ', '')) >= 1;**

* **Soru 8:**

****

**SELECT**

**FIRST\_NAME,**

**(LENGTH(FIRST\_NAME) - LENGTH(REPLACE(UPPER(FIRST\_NAME), 'E', ''))) AS E\_SAYISI**

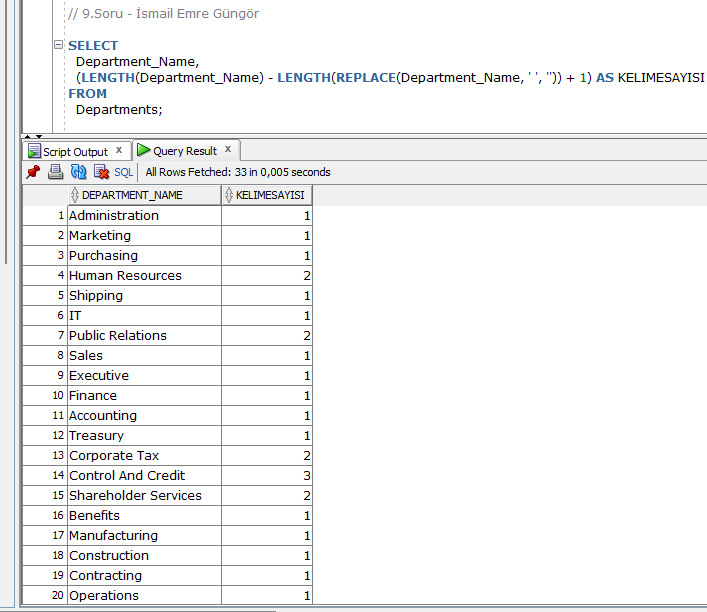
**FROM**

**Employees**

**WHERE**

**(LENGTH(FIRST\_NAME) - LENGTH(REPLACE(UPPER(FIRST\_NAME), 'E', ''))) >= 1;**

* **Soru 9:**

****

**SELECT**

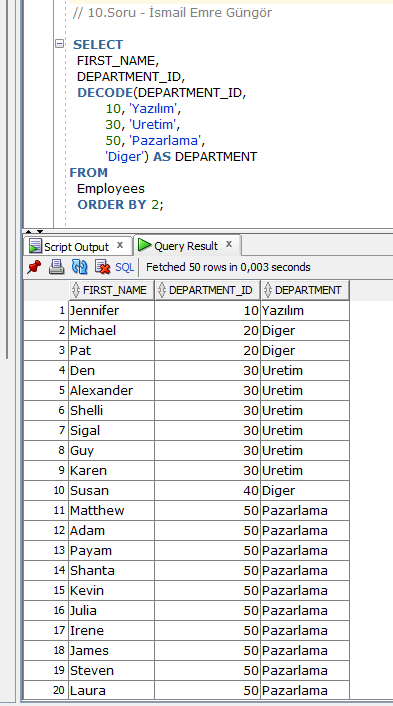
**Department\_Name,**

**(LENGTH(Department\_Name) - LENGTH(REPLACE(Department\_Name, ' ', '')) + 1) AS KELIMESAYISI**

**FROM**

**Departments;**

* **Soru 10:**



**SELECT**

**FIRST\_NAME,**

**DEPARTMENT\_ID,**

**DECODE(DEPARTMENT\_ID,**

**10, 'Yazılım',**

**30, 'Uretim',**

**50, 'Pazarlama',**

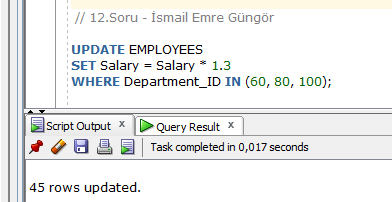
**'Diger') AS DEPARTMENT**

**FROM**

**Employees**

**ORDER BY 2;**

* **Soru 11:**
* **Count(\*):**
* Count(\*) fonksiyonu, bir sorguda döndürülen satır sayısını sayar.
* Bu sayım sırasında hiçbir sütuna özgü değerlerin varlığı veya yokluğu dikkate alınmaz, yani NULL değerler ve NOT-NULL değerler fark edilmez. Yani Count(\*), sorgunun sonucunda kaç satır döndüğünü tam olarak verir.
* **Count(commission\_pct):**
* Count(column\_name) fonksiyonu, belirli bir sütundaki NOT-NULL değerlerin sayısını sayar.
* Count(commission\_pct) özelinde, yalnızca commission\_pct sütununda NULL olmayan değerlerin sayısını hesaplar. Bu sayım sırasında NULL olan değerler göz ardı edilir ve sayılmaz.
* **Soru 12:**

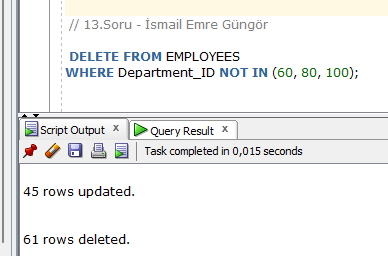
****

**UPDATE EMPLOYEES**

**SET Salary = Salary \* 1.3**

**WHERE Department\_ID IN (60, 80, 100);**

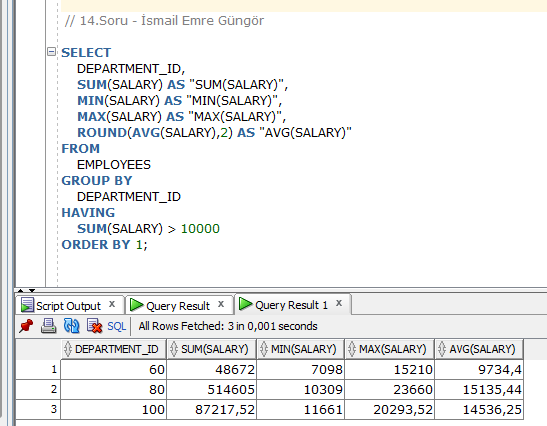
* **Soru 13:**

****

**DELETE FROM EMPLOYEES**

**WHERE Department\_ID NOT IN (60, 80, 100);**

* **Soru 14:**

****

**SELECT**

**DEPARTMENT\_ID,**

**SUM(SALARY) AS "SUM(SALARY)",**

**MIN(SALARY) AS "MIN(SALARY)",**

**MAX(SALARY) AS "MAX(SALARY)",**

**ROUND(AVG(SALARY),2) AS "AVG(SALARY)"**

**FROM**

**EMPLOYEES**

**GROUP BY**

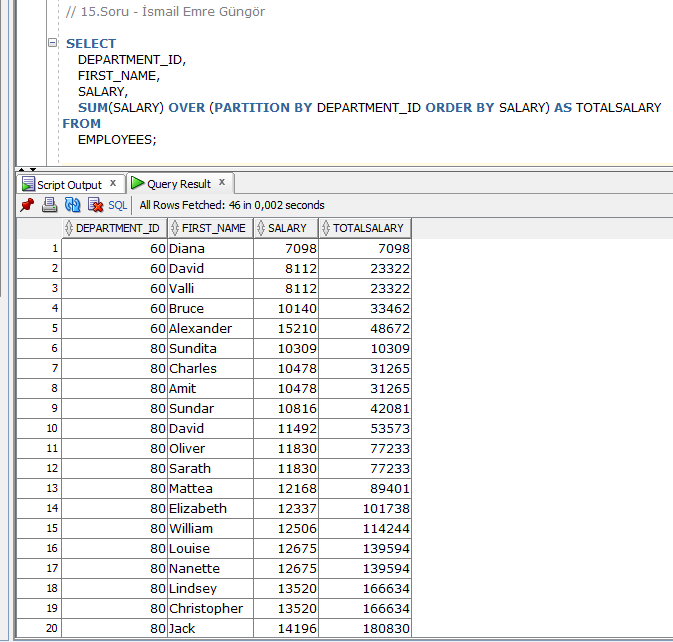
**DEPARTMENT\_ID**

**HAVING**

**SUM(SALARY) > 10000**

**ORDER BY 1;**

* **Soru 15:**

****

**SELECT**

**DEPARTMENT\_ID,**

**FIRST\_NAME,**

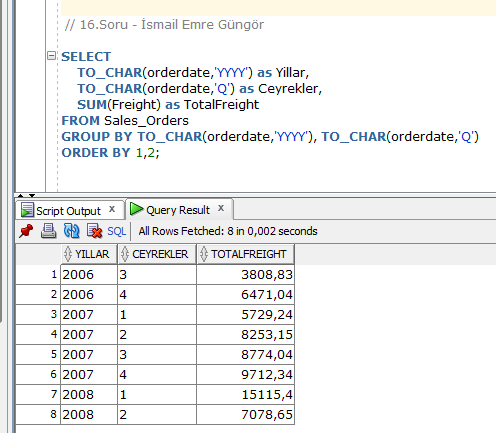
**SALARY,**

**SUM(SALARY) OVER (PARTITION BY DEPARTMENT\_ID ORDER BY SALARY) AS TOTALSALARY**

**FROM**

**EMPLOYEES;**

* **Soru 16:**

****

**SELECT**

**TO\_CHAR(orderdate,'YYYY') as Yillar,**

**TO\_CHAR(orderdate,'Q') as Ceyrekler,**

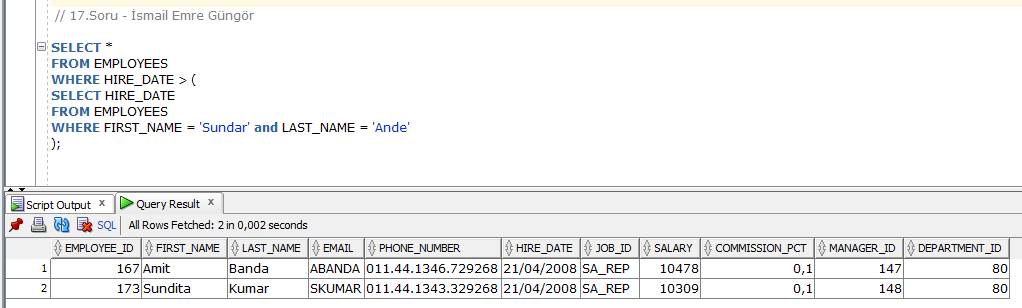
**SUM(Freight) as TotalFreight**

**FROM Sales\_Orders**

**GROUP BY TO\_CHAR(orderdate,'YYYY'), TO\_CHAR(orderdate,'Q')**

**ORDER BY 1,2;**

* **Soru 17:**

****

**SELECT \***

**FROM EMPLOYEES**

**WHERE HIRE\_DATE > (**

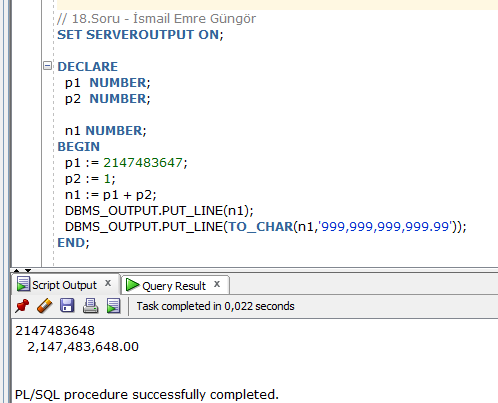
**SELECT HIRE\_DATE**

**FROM EMPLOYEES**

**WHERE FIRST\_NAME = 'Sundar' and LAST\_NAME = 'Ande'**

**);**

* **Soru 18:**
* **Hata:**
* Bir tamsayı değişkeninin maksimum kapasitesinin aşılması nedeniyle oluşan sayısal bir taşmadır (numeric overflow). PL/SQL'de PLS\_INTEGER veri tipi, 32-bit tamsayılar için kullanılır ve bu tipin alabileceği maksimum değer 2147483647'dir. p1'e bu maksimum değeri atadıktan sonra, p2'ye 1 eklemeye çalışmak bu limiti aşar ve Oracle bu işlemi gerçekleştiremez.
* **Hatanın çözüm yolu:**
* Sayıları işlem yapmadan önce NUMBER türüne dönüştürmek, böylece daha büyük değerleri işleyebilmek.



**DECLARE**

**p1 NUMBER;**

**p2 NUMBER;**

**n1 NUMBER;**

**BEGIN**

**p1 := 2147483647;**

**p2 := 1;**

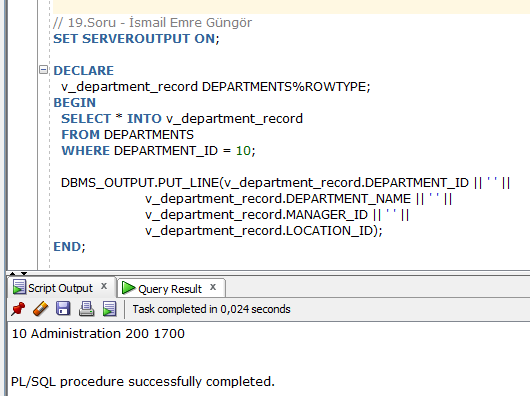
**n1 := p1 + p2;**

**DBMS\_OUTPUT.PUT\_LINE(n1);**

**DBMS\_OUTPUT.PUT\_LINE(TO\_CHAR(n1,'999,999,999,999.99'));**

**END;**

* **Soru 19:**

****

**DECLARE**

**v\_department\_record DEPARTMENTS%ROWTYPE;**

**BEGIN**

**SELECT \* INTO v\_department\_record**

**FROM DEPARTMENTS**

**WHERE DEPARTMENT\_ID = 10;**

**DBMS\_OUTPUT.PUT\_LINE(v\_department\_record.DEPARTMENT\_ID || ' ' ||**

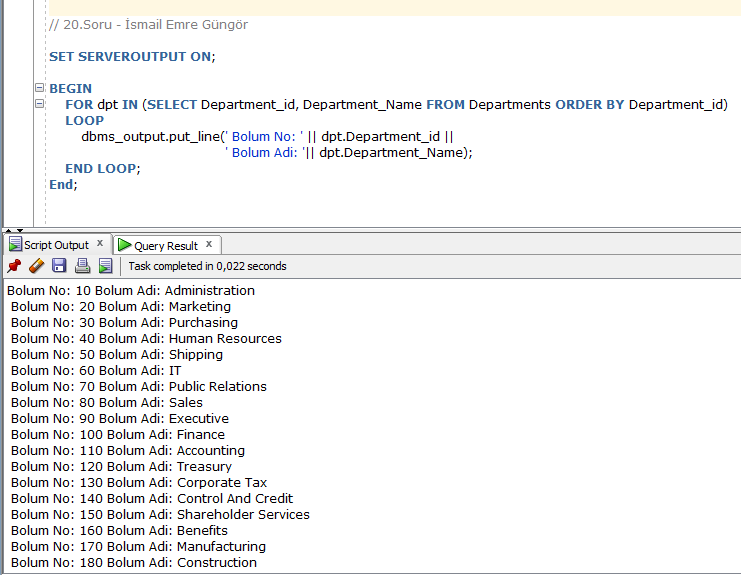
**v\_department\_record.DEPARTMENT\_NAME || ' ' ||**

**v\_department\_record.MANAGER\_ID || ' ' ||**

**v\_department\_record.LOCATION\_ID);**

**END;**

* **Soru 20:**

****

**BEGIN**

**FOR dpt IN (SELECT Department\_id, Department\_Name FROM Departments ORDER BY Department\_id)**

**LOOP**

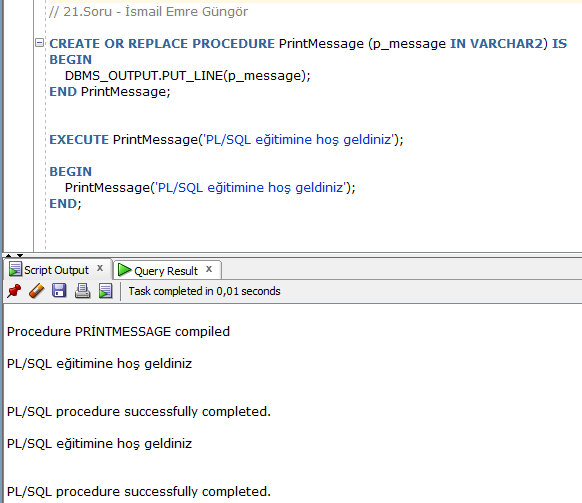
**dbms\_output.put\_line(' Bolum No: ' || dpt.Department\_id ||**

**' Bolum Adi: '|| dpt.Department\_Name);**

**END LOOP;**

**End;**

* **Soru 21:**

****

**CREATE OR REPLACE PROCEDURE PrintMessage (p\_message IN VARCHAR2) IS**

**BEGIN**

**DBMS\_OUTPUT.PUT\_LINE(p\_message);**

**END PrintMessage;**

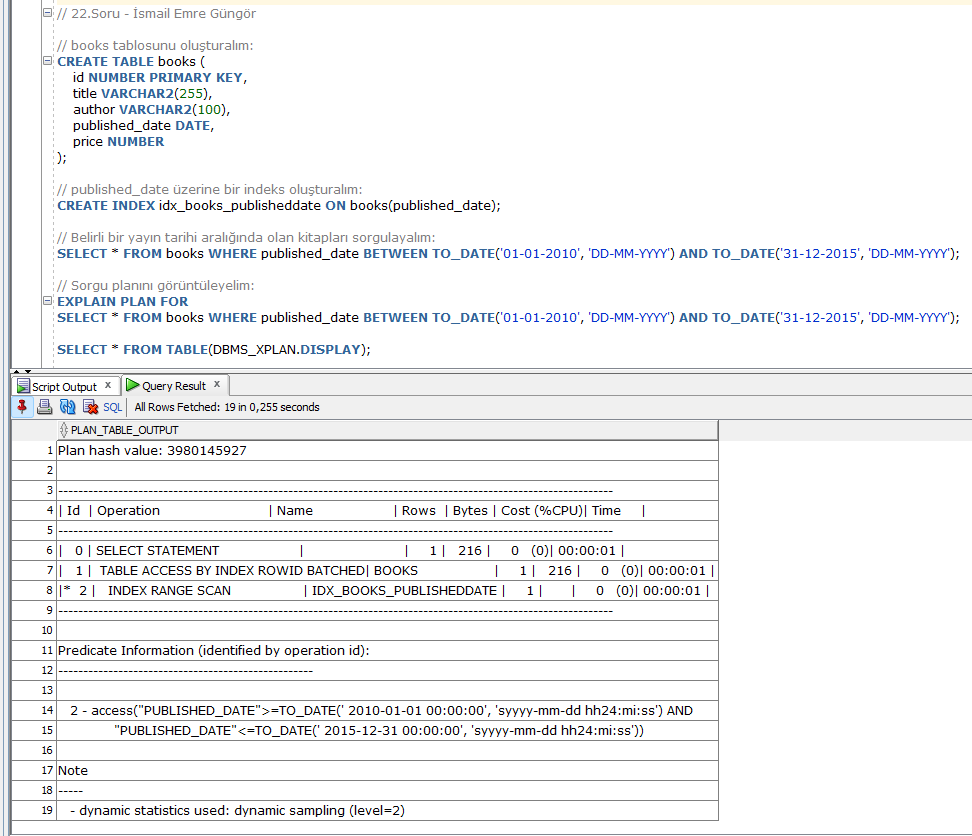
**EXECUTE PrintMessage('PL/SQL eğitimine hoş geldiniz');**

**BEGIN**

**PrintMessage('PL/SQL eğitimine hoş geldiniz');**

**END;**

* **Soru 22:**
* **Index Range Scan Nedir?**
* Index Range Scan, bir sorgunun belirli bir aralıktaki (başlangıç ve bitiş değerleri arasında kalan) kayıtları seçmesi durumunda gerçekleşen bir erişim yöntemidir.
* Sorgu optimizasyonu sırasında, sorgunun WHERE koşulunda belirtilen aralıkta yer alan verileri bulmak için indeksin kullanılmasına karar verilir.
* İndeks, arama kriterlerine göre sıralandığı için, aranan değerler indekste hızlı bir şekilde bulunabilir.

****

**// books tablosunu oluşturalım:**

**CREATE TABLE books (**

**id NUMBER PRIMARY KEY,**

**title VARCHAR2(255),**

**author VARCHAR2(100),**

**published\_date DATE,**

**price NUMBER**

**);**

**// published\_date üzerine bir indeks oluşturalım:**

**CREATE INDEX idx\_books\_publisheddate ON books(published\_date);**

**// Belirli bir yayın tarihi aralığında olan kitapları sorgulayalım:**

**SELECT \* FROM books WHERE published\_date BETWEEN TO\_DATE('01-01-2010', 'DD-MM-YYYY') AND TO\_DATE('31-12-2015', 'DD-MM-YYYY');**

**// Sorgu planını görüntüleyelim:**

**EXPLAIN PLAN FOR**

**SELECT \* FROM books WHERE published\_date BETWEEN TO\_DATE('01-01-2010', 'DD-MM-YYYY') AND TO\_DATE('31-12-2015', 'DD-MM-YYYY');**

**SELECT \* FROM TABLE(DBMS\_XPLAN.DISPLAY);**