

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 saklanması önlenerek veri tekrarı 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.

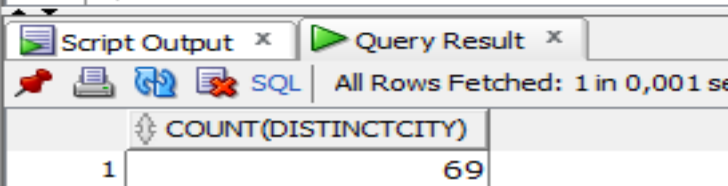
➤ **Örnek:**

```
// 91 customers kaydı var
select * from customers;

// 91 city
select
count(city)
from customers;

// 91 city
select
distinct count(city)
from customers;

// 69 city
select
count(distinct city)
from customers;
```



COUNT(DISTINCTCITY)
69

→ **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:

```
// 6.Soru - İsmail Emre Güngör
```

```
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;
```

	YILLAR	TOTALSALARY
1	2005	204490
2	2006	136383
3	2007	107217

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:

```
// 7.Soru - İsmail Emre Güngör

SELECT Department_Name
FROM Departments
WHERE LENGTH(Department_Name) - LENGTH(REPLACE(Department_Name, ' ', '')) >= 1;
```

Script Output x Query Result x

SQL All Rows Fetched: 15 in 0,002 seconds

	DEPARTMENT_NAME
1	Human Resources
2	Public Relations
3	Corporate Tax
4	Control And Credit
5	Shareholder Services
6	IT Support
7	IT Helpdesk
8	Government Sales
9	Retail Sales
10	Ulusal Arastir ve Gelistir
11	IT B
12	IT B
13	IT B
14	IT B
15	IT Space

SELECT Department_Name

FROM Departments

WHERE LENGTH(Department_Name) - LENGTH(REPLACE(Department_Name, ' ', '')) >= 1;

→ Soru 8:

// 8.Soru - İsmail Emre Güngör

```
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;
```

Script Output x Query Result x

SQL Fetched 50 rows in 0,001 seconds

	FIRST_NAME	E_SAYISI
1	Steven	2
2	Neena	2
3	Lex	1
4	Alexander	2
5	Bruce	1
6	Daniel	1
7	Ismael	1
8	Jose Manuel	2
9	Den	1
10	Alexander	2
11	Shelli	1
12	Karen	1
13	Matthew	1
14	Kevin	1
15	Irene	2
16	James	1
17	Steven	2
18	Mozhe	1
19	James	1
20	Michael	1


```
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:

// 9.Soru - İsmail Emre Güngör

```
SELECT
    Department_Name,
    (LENGTH(Department_Name) - LENGTH(REPLACE(Department_Name, ' ', '')) + 1) AS KELIMESAYISI
FROM
    Departments;
```

Script Output x Query Result x

 All Rows Fetched: 33 in 0,005 seconds

	DEPARTMENT_NAME	KELIMESAYISI
1	Administration	1
2	Marketing	1
3	Purchasing	1
4	Human Resources	2
5	Shipping	1
6	IT	1
7	Public Relations	2
8	Sales	1
9	Executive	1
10	Finance	1
11	Accounting	1
12	Treasury	1
13	Corporate Tax	2
14	Control And Credit	3
15	Shareholder Services	2
16	Benefits	1
17	Manufacturing	1
18	Construction	1
19	Contracting	1
20	Operations	1

```
SELECT
    Department_Name,
    (LENGTH(Department_Name) - LENGTH(REPLACE(Department_Name, ' ', '')) + 1) AS KELIMESAYISI
FROM
    Departments;
```

→ Soru 10:

```
// 10.Soru - İsmail Emre Güngör

SELECT
  FIRST_NAME,
  DEPARTMENT_ID,
  DECODE(DEPARTMENT_ID,
    10, 'Yazılım',
    30, 'Uretim',
    50, 'Pazarlama',
    'Diger') AS DEPARTMENT
FROM
  Employees
ORDER BY 2;
```

Script Output x Query Result x

SQL | Fetched 50 rows in 0,003 seconds

	FIRST_NAME	DEPARTMENT_ID	DEPARTMENT
1	Jennifer	10	Yazılım
2	Michael	20	Diger
3	Pat	20	Diger
4	Den	30	Uretim
5	Alexander	30	Uretim
6	Shelli	30	Uretim
7	Sigal	30	Uretim
8	Guy	30	Uretim
9	Karen	30	Uretim
10	Susan	40	Diger
11	Matthew	50	Pazarlama
12	Adam	50	Pazarlama
13	Payam	50	Pazarlama
14	Shanta	50	Pazarlama
15	Kevin	50	Pazarlama
16	Julia	50	Pazarlama
17	Irene	50	Pazarlama
18	James	50	Pazarlama
19	Steven	50	Pazarlama
20	Laura	50	Pazarlama

```
SELECT
  FIRST_NAME,
  DEPARTMENT_ID,
  DECODE(DEPARTMENT_ID,
    10, 'Yazılım',
    30, 'Uretim',
    50, 'Pazarlama',
    'Diger') AS DEPARTMENT
FROM
  Employees
ORDER BY 2;
```

→ **Soru 11:**

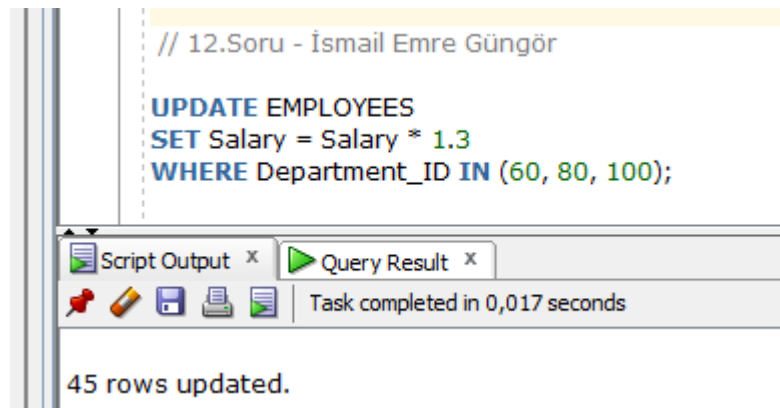
➤ **Count(*):**

- 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):**

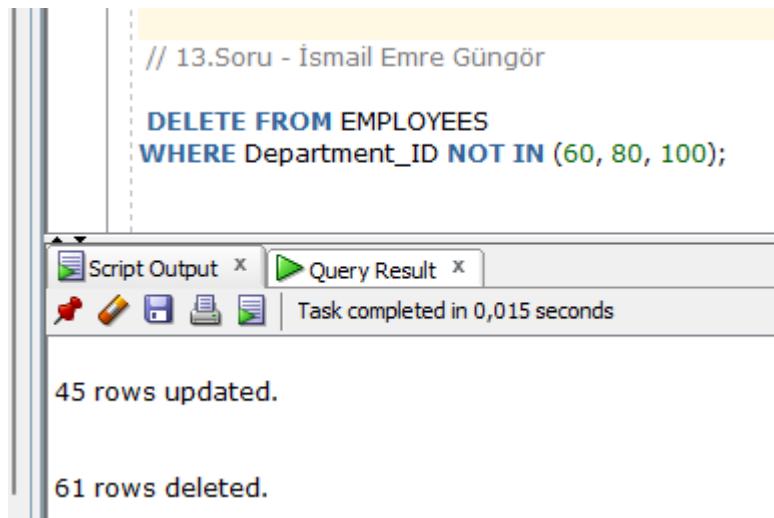
- Count(column_name) fonksiyonu, belirli bir sütundaki NOT-NULL değerlerin sayısını sayar.
- Count(commission_pct) özelinde, yalnızca commission_pct sütununda NULL olmayan değerlerin sayısını hesaplar. Bu sayım sırasında NULL olan değerler göz ardı edilir ve sayılmaz.

→ 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:

```
// 14.Soru - İsmail Emre Güngör

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;
```

Script Output x Query Result x Query Result 1 x

SQL All Rows Fetched: 3 in 0,001 seconds

	DEPARTMENT_ID	SUM(SALARY)	MIN(SALARY)	MAX(SALARY)	AVG(SALARY)
1	60	48672	7098	15210	9734,4
2	80	514605	10309	23660	15135,44
3	100	87217,52	11661	20293,52	14536,25

```
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:

// 15.Soru - İsmail Emre Güngör

```
SELECT
DEPARTMENT_ID,
FIRST_NAME,
SALARY,
SUM(SALARY) OVER (PARTITION BY DEPARTMENT_ID ORDER BY SALARY) AS TOTALSALARY
FROM
EMPLOYEES;
```

Script Output x Query Result x				
All Rows Fetched: 46 in 0,002 seconds				
	DEPARTMENT_ID	FIRST_NAME	SALARY	TOTALSALARY
1	60	Diana	7098	7098
2	60	David	8112	23322
3	60	Valli	8112	23322
4	60	Bruce	10140	33462
5	60	Alexander	15210	48672
6	80	Sundita	10309	10309
7	80	Charles	10478	31265
8	80	Amit	10478	31265
9	80	Sundar	10816	42081
10	80	David	11492	53573
11	80	Oliver	11830	77233
12	80	Sarath	11830	77233
13	80	Mattea	12168	89401
14	80	Elizabeth	12337	101738
15	80	William	12506	114244
16	80	Louise	12675	139594
17	80	Nanette	12675	139594
18	80	Lindsey	13520	166634
19	80	Christopher	13520	166634
20	80	Jack	14196	180830

```
SELECT
DEPARTMENT_ID,
FIRST_NAME,
SALARY,
SUM(SALARY) OVER (PARTITION BY DEPARTMENT_ID ORDER BY SALARY) AS TOTALSALARY
FROM
EMPLOYEES;
```

→ Soru 16:

// 16.Soru - İsmail Emre Güngör

```
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;
```

	YILLAR	CEYREKLER	TOTALFREIGHT
1	2006	3	3808,83
2	2006	4	6471,04
3	2007	1	5729,24
4	2007	2	8253,15
5	2007	3	8774,04
6	2007	4	9712,34
7	2008	1	15115,4
8	2008	2	7078,65

```
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:

```
// 17.Soru - İsmail Emre Güngör

SELECT *
FROM EMPLOYEES
WHERE HIRE_DATE > (
SELECT HIRE_DATE
FROM EMPLOYEES
WHERE FIRST_NAME = 'Sundar' and LAST_NAME = 'Ande'
);
```

Script Output x Query Result x

All Rows Fetched: 2 in 0,002 seconds

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	167	Amit	Banda	ABANDA	011.44.1346.729268	21/04/2008	SA_REP	10478	0,1	147	80
2	173	Sundita	Kumar	SKUMAR	011.44.1343.329268	21/04/2008	SA_REP	10309	0,1	148	80

```
SELECT *
FROM EMPLOYEES
WHERE HIRE_DATE > (
SELECT HIRE_DATE
FROM EMPLOYEES
WHERE FIRST_NAME = 'Sundar' and LAST_NAME = 'Ande'
);
```

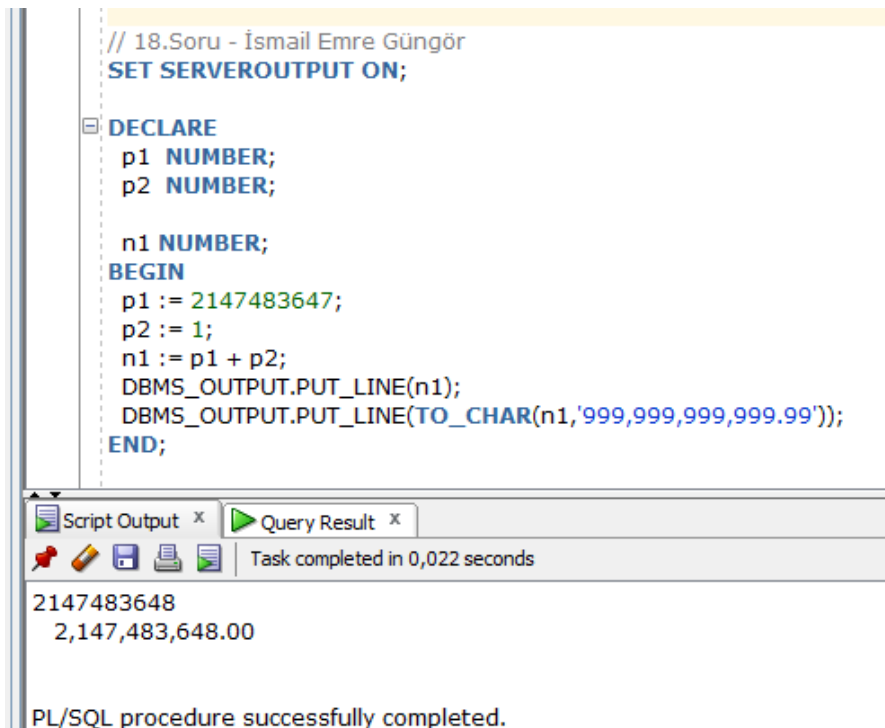
→ Soru 18:

➤ 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.



```
// 18.Soru - İsmail Emre Güngör
SET SERVEROUTPUT ON;

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;
```

Script Output x Query Result x

Task completed in 0,022 seconds

2147483648
2,147,483,648.00

PL/SQL procedure successfully completed.

DECLARE

p1 NUMBER;

p2 NUMBER;

n1 NUMBER;

BEGIN

p1 := 2147483647;

p2 := 1;

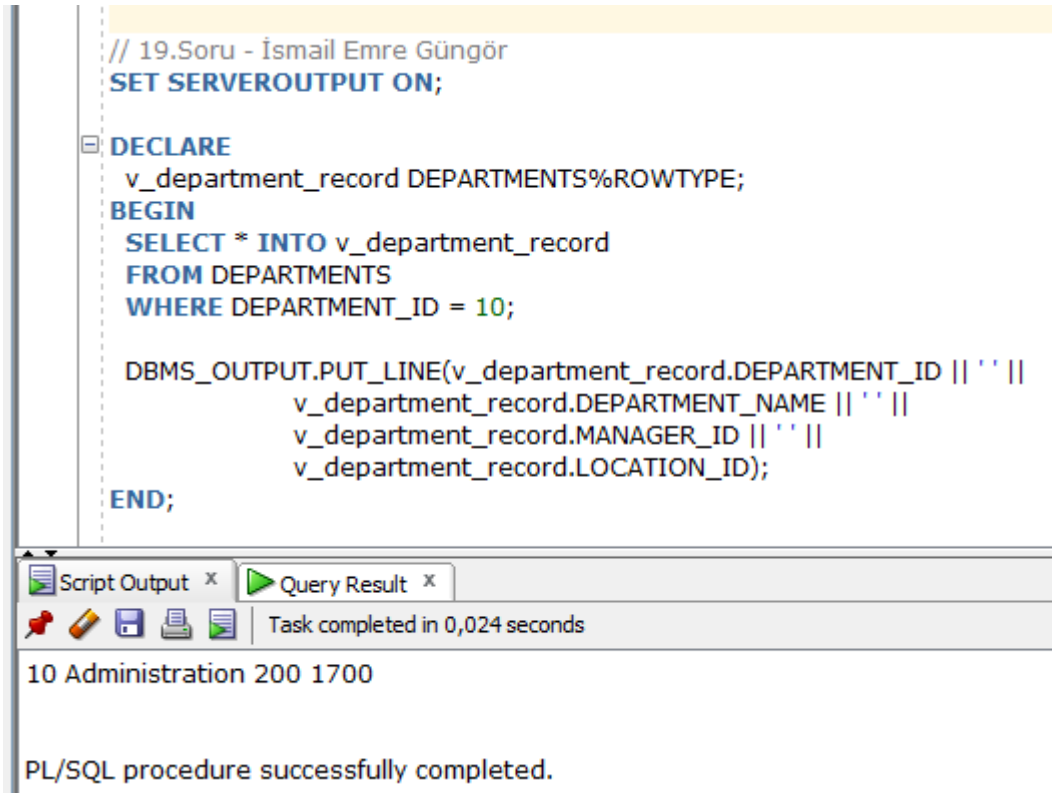
n1 := p1 + p2;

DBMS_OUTPUT.PUT_LINE(n1);

DBMS_OUTPUT.PUT_LINE(TO_CHAR(n1,'999,999,999,999.99'));

END;

→ Soru 19:



The screenshot shows the Oracle SQL Developer interface. The main editor window contains a PL/SQL script for a procedure named '19.Soru - İsmail Emre Güngör'. The script sets server output on, declares a record variable, selects data from the DEPARTMENTS table for department ID 10, and prints the details using DBMS_OUTPUT.PUT_LINE. Below the editor, the 'Script Output' and 'Query Result' tabs are visible. The 'Script Output' tab shows the message 'Task completed in 0,024 seconds'. The 'Query Result' tab displays the output of the procedure: '10 Administration 200 1700'. At the bottom, a status bar indicates 'PL/SQL procedure successfully completed.'

```
// 19.Soru - İsmail Emre Güngör
SET SERVEROUTPUT ON;

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;
```

Script Output x Query Result x

Task completed in 0,024 seconds

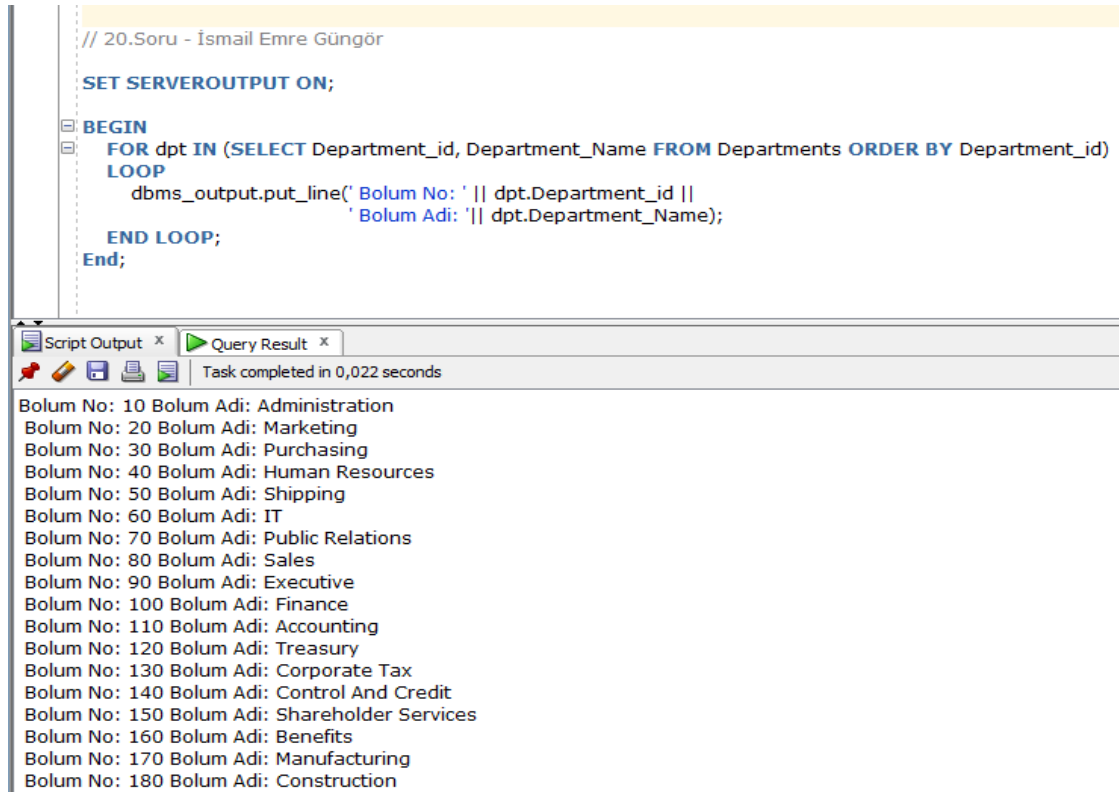
10 Administration 200 1700

PL/SQL procedure successfully completed.

```
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:



The screenshot shows the Oracle SQL Developer interface. The top pane displays a PL/SQL script for a loop that iterates over all departments in the 'Departments' table, ordered by department ID. The script uses the 'dbms_output.put_line' procedure to print the department ID and name. The bottom pane shows the 'Script Output' tab, which displays the results of the script execution. The output consists of 18 lines, each showing a department ID and its name, separated by a vertical bar.

```
// 20.Soru - İsmail Emre Güngör

SET SERVEROUTPUT ON;

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;
```

Script Output x Query Result x

Task completed in 0,022 seconds

Bolum No: 10 Bolum Adi: Administration
Bolum No: 20 Bolum Adi: Marketing
Bolum No: 30 Bolum Adi: Purchasing
Bolum No: 40 Bolum Adi: Human Resources
Bolum No: 50 Bolum Adi: Shipping
Bolum No: 60 Bolum Adi: IT
Bolum No: 70 Bolum Adi: Public Relations
Bolum No: 80 Bolum Adi: Sales
Bolum No: 90 Bolum Adi: Executive
Bolum No: 100 Bolum Adi: Finance
Bolum No: 110 Bolum Adi: Accounting
Bolum No: 120 Bolum Adi: Treasury
Bolum No: 130 Bolum Adi: Corporate Tax
Bolum No: 140 Bolum Adi: Control And Credit
Bolum No: 150 Bolum Adi: Shareholder Services
Bolum No: 160 Bolum Adi: Benefits
Bolum No: 170 Bolum Adi: Manufacturing
Bolum No: 180 Bolum Adi: Construction

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:

// 21.Soru - İsmail Emre Güngör

```
CREATE OR REPLACE PROCEDURE PrintMessage (p_message IN VARCHAR2) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(p_message);
END PrintMessage;

EXECUTE PrintMessage('PL/SQL eğitime hoş geldiniz');

BEGIN
    PrintMessage('PL/SQL eğitime hoş geldiniz');
END;
```

Script Output x Query Result x
Task completed in 0,01 seconds

Procedure PRINTMESSAGE compiled

PL/SQL eğitime hoş geldiniz

PL/SQL procedure successfully completed.

PL/SQL eğitime hoş geldiniz

PL/SQL procedure successfully completed.

```
CREATE OR REPLACE PROCEDURE PrintMessage (p_message IN VARCHAR2) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(p_message);
END PrintMessage;
```

```
EXECUTE PrintMessage('PL/SQL eğitime hoş geldiniz');
```

```
BEGIN
    PrintMessage('PL/SQL eğitime 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 aralıktaki yer alan verileri bulmak için indeksin kullanılmasına karar verilir.
- İndeks, arama kriterlerine göre sıralandığı için, aranan değerler indekste hızlı bir şekilde bulunabilir.

```
// 22.Soru - İsmail Emre Güngör

// 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 yayının 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);
```

Script Output x Query Result x

All Rows Fetched: 19 in 0,255 seconds

PLAN_TABLE_OUTPUT							
1	Plan hash value: 3980145927						
2							
3	-----						
4	Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
5	-----						
6	0	SELECT STATEMENT		1	216	0 (0)	00:00:01
7	1	TABLE ACCESS BY INDEX ROWID BATCHED	BOOKS	1	216	0 (0)	00:00:01
8	* 2	INDEX RANGE SCAN	IDX_BOOKS_PUBLISHEDDATE	1		0 (0)	00:00:01
9	-----						
10							
11	Predicate Information (identified by operation id):						
12	-----						
13							
14	2	access("PUBLISHED_DATE">=TO_DATE(' 2010-01-01 00:00:00', 'yyyy-mm-dd hh24:mi:ss') AND					
15		"PUBLISHED_DATE"<=TO_DATE(' 2015-12-31 00:00:00', 'yyyy-mm-dd hh24:mi:ss'))					
16							
17	Note						
18	-----						
19	- dynamic statistics used: dynamic sampling (level=2)						

// books tablosunu oluřturalım:

```
CREATE TABLE books (  
    id NUMBER PRIMARY KEY,  
    title VARCHAR2(255),  
    author VARCHAR2(100),  
    published_date DATE,  
    price NUMBER  
);
```

// published_date üzerine bir indeks oluřturalım:

```
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);
```