veritabanı final

Hafta 1

- İşlenerek anlam kazandırılmamış ham gerçeklere veri denir.
- Veriler işlenerek bilgi oluşturulur
- Bilgi, verinin anlamını göstermek için kullanılır.
- Dosyalarda depolanan birbiriyle ilişkili veri topluluklarına veritabanı denir .
    - Veritabanı = HamVeri + ÜstVeri/Metadata (İlişkiler+Veri Karakteristikleri)

VTYS ile Dosya Sisteminin Karşılaştırılması

- **Veri Tümleştirme (Data Integration):** Verilerin tekrarsız olarak saklanması.
- **Veri Bütünlüğü (Data Integrity):** Verilerin bozulmadan ve tutarlı olarak saklanması sağlanabilir. Kısıtlar eklenerek veri tutarsızlığı önlenebilir (key constraints, integrity rules).
- **Veri Güvenliği (Data Security):** Sistem hataları karşısında ya da saldırıya rağmen verilerin kaybolmaması ve tutarlılığının korunması sağlanabilir (transaction, raid sistemler, kurtarma mekanizmaları, gelişmiş yetkilendirme yapısı vb.).
- **Veri Soyutlama (Data Abstraction):** Kullanıcıya, karmaşık yapıdaki fiziksel veri yapısı yerine anlaşılabilirliği ve yönetilebilirliği daha kolay olan mantıksal model sunulur.

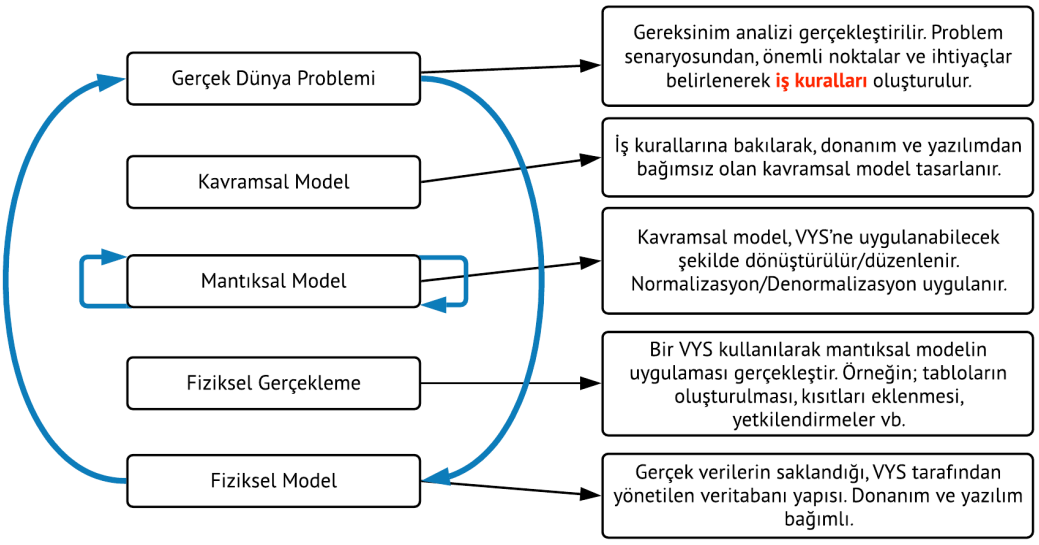
Veritabanı Kullanım Amaçları

Operasyonel: Veriler üzerinde sürekli değişiklikler yapılır.

Veri Ambarı: Veriler raporlama ve karar destek amaçlarıyla kullanılır.

Hafta 2

Veritabanı Geliştirme Yaşam Döngüsü



İş Kuralları (Business Rules)

•

İş kuralı: Veritabanı tasarımı yapılacak organizasyon ile ilgili işleyiş, kural ya da yönetmeliğin özetlenmiş şekline iş kuralları denilebilir. İş kuralları ihtiyaç listesine benzer.

- Bir müşteri çok sayıda sipariş verebilir.

Hafta 6

Yapısal Sorgulama Dili (SQL)

- Veri Tanımlama Dili (Data Definition Language, DDL)
  - Yapısal Komutlar
  - Veritabanı, tablo, ilişki vs. oluşturma, değiştirme, silme vs.
- Veri İşleme Dili (Data Manipulation Language, DML)
  - Veri ekleme, silme, güncelleme, sorgulama vs.

Temel SQL Komutları

SELECT

```
SELECT * FROM "customers";
```

WHERE

- İstenilen koşula uyan kayıtların listelenmesi için WHERE komutu kullanılır.

```
SELECT * FROM "customers" WHERE "Country" = 'Argentina';
```

DISTINCT

- Sorgu sonucunda yer alan tekrarlı kayıtların (satırların), tek kayıt olarak getirilmesini sağlar.

```
Kaç farklı şehirden müşteri bulunmaktadır?
SELECT DISTINCT "City" from "customers";
```

ORDER BY

- Sorgular sonucunda listelenen kayıtların belirli alanlara göre alfabetik veya sayısal olarak artan ya da azalan şeklinde sıralanması için

`SELECT * FROM "customers" ORDER BY "ContactName" ASC;`

LIKE / NOT LIKE

`SELECT * FROM "customers" WHERE "Country" LIKE 'P%';` #P ile başla

`SELECT * FROM "customers" WHERE "Country" NOT LIKE 'P%'; # NULL olanlar getirilmez` #P ile başlama

`SELECT * FROM "customers" WHERE "Country" LIKE '%e';` #e ile bit

`SELECT * FROM "customers" WHERE "Country" LIKE '_a%';2. harf a`

`SELECT * FROM "customers" WHERE "Country" LIKE '%pa%';` #pa barındır

`SELECT * FROM "customers" WHERE "Country" LIKE '%pa_';` #pa ile başla

BETWEEN

`SELECT * FROM "products" WHERE "UnitPrice" BETWEEN 10 AND 20;`

IN

```
SELECT * FROM "customers"WHERE "public"."customers"."Countr
y" IN ('Türkiye', 'Kuzey Kıbrıs Türk Cumhuriyeti');
```

AS

- AS ifadesi ile alanlara/tablolara takma isim verilir.

```
SELECT "CompanyName" AS "musteriler" FROM "customers";
```

NULL

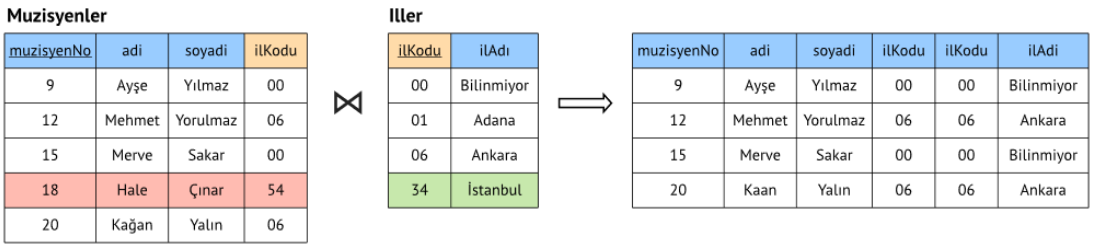
```
SELECT * FROM "customers" WHERE "Region" IS NOT NULL;
```

Tablo Birleştirme İşlemleri

Doğal/İç Birleştirme (Natural/Inner Join)

- "INNER JOIN" yerine "JOIN" ifadesi de kullanılabilir.
- **ikisinde de ortak il kodu olan değerleri tablolar**

```
SELECT * FROM "Muzisyenler" INNER JOIN "Iller"
ON "Muzisyenler"."ilKodu" = "Iller"."ilKodu"
```

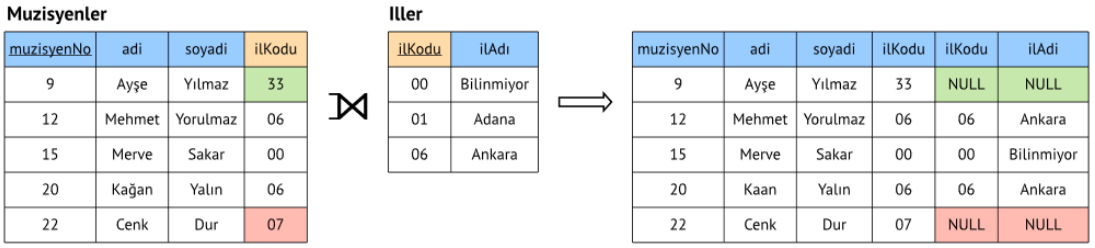


Sol Dış Birleştirme (Left Outer Join)

- **solda bulunan değerlerin hepsini getirmek zorundadır sağdaki tabloda bulunmuyorsa null yazar**

```
SELECT * FROM "Muzisyenler" LEFT OUTER JOIN "Iller"
```

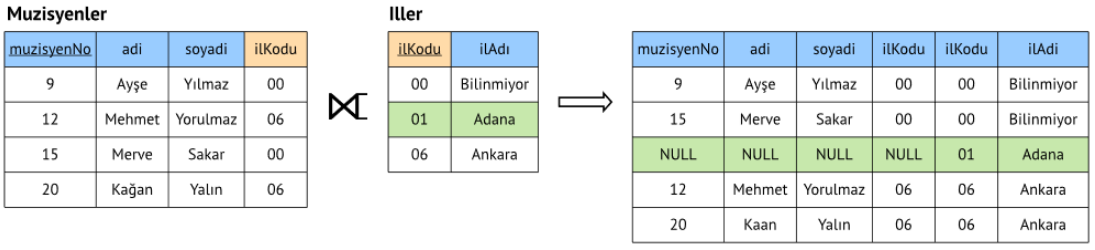
```
ON "Muzisyenler"."ilKodu" = "Iller"."ilKodu"
```



Sağ Dış Birleştirme (Right Outer Join)

- **Sağdaki tablonun tüm değerleri gelmelidir eğer solda yoksa null döndürür**

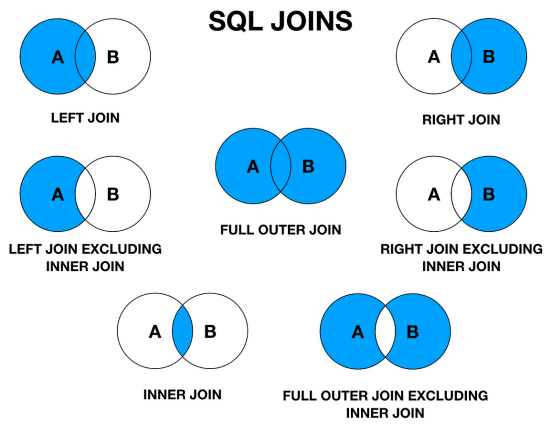
```
SELECT * FROM "Muzisyenler" RIGHT OUTER JOIN "Iller"
ON "Muzisyenler"."ilKodu" = "Iller"."ilKodu"
```

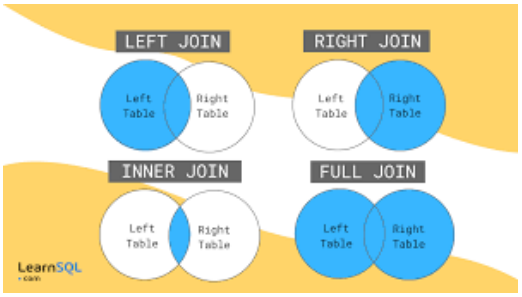


Cross Join Full outer Join

iki veya daha fazla tablonun Kartezyen ürününü döndüren benzersiz bir birleştirme işlemidir . Bu, sol tablodaki her satırı sağ tablodaki her satırla eşleştirdiği ve bunun sonucunda tüm olası kayıt çiftlerinin bir kombinasyonunu oluşturduğu anlamına gelir.

The `FULL OUTER JOIN` keyword returns all records when there is a match in left (table1) or right (table2) table records.





SELECT ... INTO

- Bir tablodan alınan verileri, yeni bir tabloya kopyalamak için kullanılır.
- **Yeni tablonun mevcut olmaması gerekir.**

```
SELECT "CompanyName", "ContactName" INTO "MusteriYedek" FROM "customers";
```

INSERT

- INSERT komutu tabloya yeni kayıt eklemek için kullanılır.

INSERT INTO () VALUES ()

UPDATE

UPDATE “” SET

```
UPDATE "customers" SET "ContactName" = 'Mario Pontes', "City"
WHERE "CompanyName" = 'Familia Arquibaldo';
//"customers" tablosunda, "CompanyName" değeri 'Familia Arqui
//olan müşteri kaydının "ContactName" değerini 'Mario Pontes'
//ve "City" değerini 'Rio de Janeiro' olarak günceller
```

- WHERE ifadesi kullanılmazsa tüm satırlar değiştirilir.

DELETE

```
DELETE FROM "customers"
WHERE "CompanyName" = 'LINO-Delicateses' AND "ContactName" =
```

Hafta 7

Temel SQL

DROP

```
DROP TABLE "sema1"."Urunler";
"sema1"."Urunler": Bu, "sema1" adlı şemada bulunan "Urunler"
 adlı tabloyu siler
```

- Bir **şema , tablolar, tetikleyiciler, saklı yordamlar vb. gibi veritabanı nesnelerinin bir koleksiyonudur. Bir şema , şema sahibi** olarak bilinen bir kullanıcıyla bağlantılıdır . Veritabanında bir veya daha fazla şema olabilir.

TRUNCATE TABLE

Bir tablonun içindeki tüm verileri silmek için kullanılır. Veriler gerçek anlamda silinir ve böylece işgal edilen yer sistem tarafından kullanılabilir hale gelir (Vacuum).

DROP

- **Amaç**: Bir veritabanı nesnesini tamamen silmek (tablo, şema, veritabanı vb.)

Etki Alanı: `DROP` komutu, tabloyu ve tüm verileri siler, ayrıca tablo yapısını (yani, tabloyu oluşturan şemayı, indeksleri, anahtarları vb.) da siler.

TRUNCATE

- **Amaç**: Bir tablodaki tüm verileri silmek, ancak tabloyu ve yapısını korumak.

Etki Alanı: `TRUNCATE` komutu, yalnızca tablodaki **verileri** siler. Tablo yapısı, indeksler, anahtarlar gibi diğer öğeler korunur. `DELETE` komutundan daha verimlidir, çünkü her bir satırı ayrı ayrı silmek yerine, tabloyu tamamen boşaltır.

ALTER TABLE

```
ALTER TABLE "Urunler" ADD COLUMN "uretimYeri" VARCHAR(30);
```

SEQUENCE

```
    "urunNo" SERIAL,->otomatik artım
//serial kullanmadan
    "urunNo" INTEGER,
CREATE SEQUENCE "sayac";
ALTER SEQUENCE "sayac" OWNED BY "Urunler"."urunNo";
VALUES
(NEXTVAL('"public"."sayac"')// eklerken böyle eklenir
SELECT NEXTVAL('sayac');
```

SQL Kısıtları (CONSTRAINTS)

UNIQUE

```
    CONSTRAINT "urunlerUnique" UNIQUE("kodu", "adi"),
kodu ve adi unique olur
ALTER TABLE "Urunler" DROP CONSTRAINT "urunlerUnique";
kısıtı siler
```

CHECK

```
    CONSTRAINT "urunlerCheck" CHECK("miktari" >= 0)
```

`CHECK` Kısıtlaması

Tablonun oluşturulması sırasında veya var olan tabloya kısıtlama

`WHERE` Koşulu

Veri sorgulama veya güncelleme işlemleri sırasında kullanılır. Veri

eklerken kullanılır. Verilerin doğruluğunu sağlamak için kullanılır.

işlemlerinde belirli kriterlere uyan satırları hedef almak için kullanılır.

Birincil Anahtar (PRIMARY KEY)

```
    CONSTRAINT "urunlerPK" PRIMARY KEY("urunNo")
ALTER TABLE "Urunler" DROP CONSTRAINT "urunlerPK";
İki alanlı birincil anahtar örneği.
    CONSTRAINT "urunlerPK1" PRIMARY KEY("urunNo", "kodu")
```

Yabancı Anahtar (FOREIGN KEY)

```
    CONSTRAINT "urunSinifiFK1" FOREIGN KEY("sinifi") REFERE
NCES "UrunSinifi"("sinifNo")
//Bu ifade yukarıdaki ile aynıdır. ON DELETE ve ON UPDATE d
urumunda
 //ne yapılacağı belirtilmediğinde varsayılan olarak NO ACT
ION olur.
    CONSTRAINT "urunSinifiFK1" FOREIGN KEY("sinifi") REFERE
NCES "UrunSinifi"("sinifNo") ON DELETE NO ACTION ON UPDATE
NO ACTION
```

Davranış Şekli	Açıklama	Etki
NO ACTION	İşlem tamamlandıktan sonra referans bütünlüğü kontrol edilir, ihlal varsa işlem başarısız olur.	Varsayılan, işlem sonrası kontrol.
RESTRICT	İşlem sırasında referans bütünlüğü kontrol edilir, ihlal varsa işlem durdurulur.	Hemen kısıtlar, işlem sırasında kontrol.
CASCADE	Referans edilen kayıt silindiğinde veya güncellendiğinde, ilişkili kayıtlar da silinir/güncellenir.	Bağlı kayıtlar otomatik olarak silinir veya güncellenir.
SET NULL	Referans edilen kayıt silindiğinde veya güncellendiğinde, ilişkili sütunlar `NULL` olur.	Bağlı sütunlar `NULL` olarak güncellenir.
SET DEFAULT	Referans edilen kayıt silindiğinde veya güncellendiğinde, ilişkili sütunlar varsayılan değere ayarlanır.	Bağlı sütunlar varsayılan değer ile güncellenir.

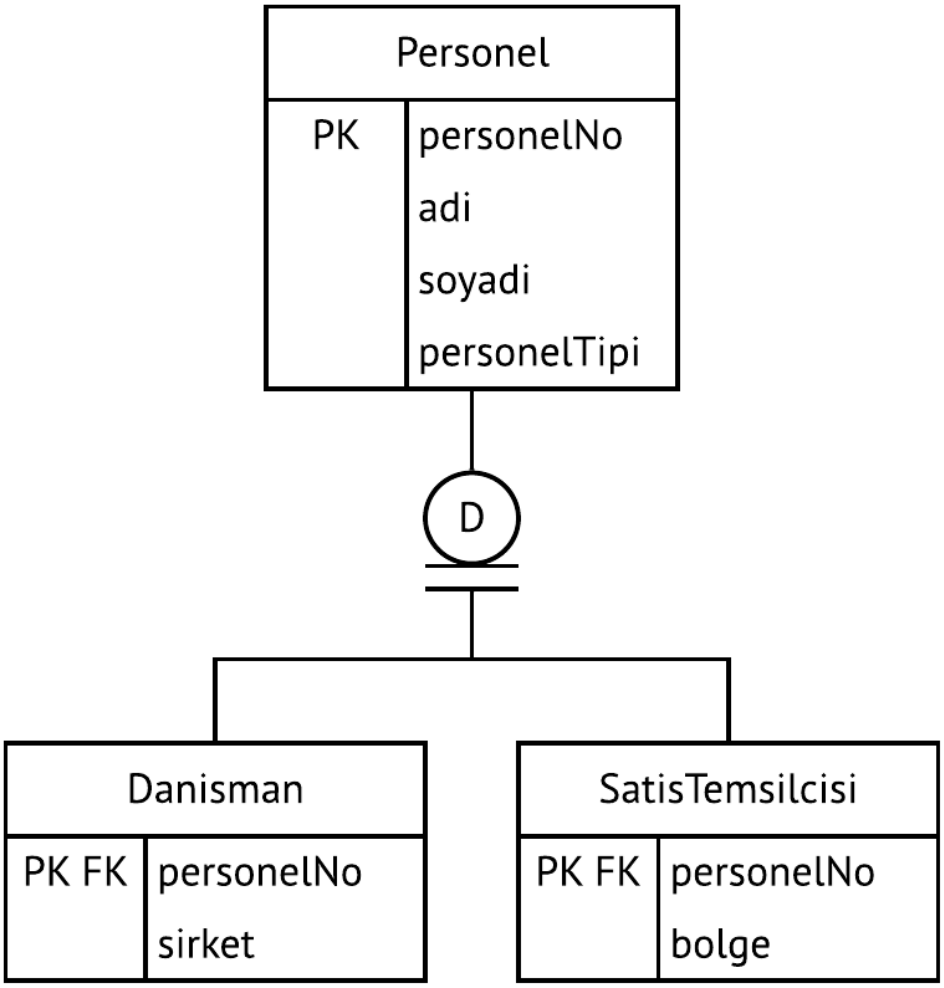
Hafta 8

İndeks

indeks (index), veritabanındaki tablo veya tablolar üzerindeki verilerin daha hızlı erişilmesini sağlamak için kullanılan bir veritabanı nesnesidir. İndeksler, bir veya daha fazla sütun üzerinde oluşturularak sorguların performansını artırır.

```
CREATE INDEX "musterilerAdiIndex" ON "Musteriler" ("adi");
```

Kalıtım



- Temel tablo ile çocuk tablo arasında bağıntı kurulumu. "CASCADE" kullanımının en uygun olduğu yer
- CASCADE **specifies that when a referenced row is deleted, row(s) referencing it should be automatically deleted as well**.

```
ALTER TABLE "personel"."Danisman"
ADD CONSTRAINT "DanismanPersonel" FOREIGN KEY ("personelNo")
REFERENCES "personel"."Personel" ("personelNo")
ON DELETE CASCADE
ON UPDATE CASCADE;
```

Tekli Bağıntı / Özyineli Birleştirme

bir tablonun kendisi ile birleştirilmesi işlemidir. Bu, tabloda bulunan kayıtların, aynı tabloda bulunan diğer kayıtlarla ilişkilendirilmesine olanak tanır.

```
INSERT INTO "Personel" ("adi", "soyadi") VALUES ('Ahmet', 'Şa
INSERT INTO "Personel" ("adi", "soyadi") VALUES ('Ayşe', 'Kar
INSERT INTO "Personel" ("adi", "soyadi", "yoneticisi") VALUES
INSERT INTO "Personel" ("adi", "soyadi", "yoneticisi") VALUES
İlk iki satırda, Ahmet ve Ayşe adında yöneticisi olmayan çalı
eklenmiştir.
Üçüncü ve dördüncü satırlarda, Mustafa ve Fatma adında yöneti
Ahmet ve Ayşe olan çalışanlar eklenmiştir.
```

Görünüm (View)

- Bir veya daha fazla tablodan seçilen satırlar ve alanlardaki bilgilerin yeni bir tablo gibi ele alınmasını temin eden yapıdır.
- Dinamiktir. GÖRÜNÜM (VIEW) ile oluşturulan tabloya gerçekleştirilen her erişimde kendisini oluşturan ifadeler (görünüm – view ifadeleri) yeniden çalıştırılır.
  • Örneğin şirket personeli, müşterilerin genel bilgilerini (ad, soyad, adres v.b.) görebilsin ancak kredi kartı bilgilerine erişemesin isteniyorsa yalnızca görmesini istediğimiz bilgileri içeren bir görünüm oluşturulabilir ve ilgili personeli bu görünüme yetkilendiririz.

Çoklu Satır Fonksiyonları

COUNT

• Sorgu sonucunda oluşan sonuç kümesindeki satır sayısını döndürür.
• Yalnızca bir sütun için uygulanırsa o sütundaki NULL olmayan kayıtların sayısı bulunur.

MAX

- Seçilen sütundaki en büyük değere ulaşmak için kullanılır.

AVG

- Seçilen sütundaki değerlerin aritmetik ortalamasını bulmak için kullanılır.

LIMIT

```
SELECT * FROM "products" ORDER BY "ProductID" ASC LIMIT 4;
```

Bu sorgu,

`"products"` tablosundan tüm sütunları seçip, sonuçları `"ProductID"` sütununa göre artan (ascending) sırayla sıralar ve yalnızca ilk 4 satırı döndürür.

GROUP BY

• Sorgu sonucunu belirtilen alan(lar)a göre gruplayarak oluşturur.

```
SELECT "SupplierID", COUNT("SupplierID") AS "urunSayisi"
FROM "products"
GROUP BY "SupplierID";
SELECT "SupplierID", SUM("UnitsInStock") AS "stokSayisi"
FROM "products"
GROUP BY "SupplierID";
 ürünleri tedarikçilerine (SupplierID) göre gruplar ve
 her tedarikçinin sağladığı ürünlerin sayısını (COUNT)
 hesaplayarak tedarikçi bilgisi ile birlikte döndürür.
```

!HAVING

• Gruplandırılmış veriler üzerinde filtreleme yapmak için kullanılır.

•

Gruplama işleminden sonra koşul yazılabilmesi için WHERE yerine HAVING ifadesinin kullanılması gereklidir.

```
SELECT "SupplierID", COUNT("SupplierID") AS "urunSayisi"FRO
M
"products"GROUP BY "SupplierID"HAVING COUNT("SupplierID") >
2

"products" tablosundan her "SupplierID" için ürün sayısını
gruplar ve bu sayının 2'den büyük olduğu tedarikçileri döne
r.
```

- **COUNT("SupplierID") > 2** ifadesi, **SupplierID** başına kaç ürün olduğunu sayarak belirli bir gruplama üzerinde çalışır.
- **WHERE** ifadesi, bu tür **aggregated** (toplanmış) verilere doğrudan uygulanamaz çünkü **WHERE** ifadesi, gruplama işleminden **önce** satırları filtreler.
- **Çoklu satır fonksiyonları ile WHERE kullanılmaz.**

Hafta 10

Fonksiyon (Function) / Saklı Yordam (Stored Procedure)

Avantajları

- bir defa oluşturulduktan sonra derlenerek sistem kataloğunda saklanır. Her çağrıldıklarında SQL motoru tarafından derlenmek zorunda olan SQL ifadelerine göre çok daha hızlıdır.
- Uzun SQL ifadeleri yerine fonksiyonun / saklı yordamın adını ve parametrelerini göndermek yeterlidir
- Yeniden kullanılabilir (reusable).
- Veritabanı yöneticisi, fonksiyonlara / saklı yordamlara hangi uygulamalar tarafından erişileceğini, tabloların güvenlik düzeyleriyle uğraşmadan, kolayca belirleyebilir.

Dezavantajları

- uygulama ile veritabanı arasındaki bağımlılık artar ve veritabanından bağımsız kodlama yapmak gitgide imkansızlaşır.
- Fonksiyonların / saklı yordamların yapacağı işler uygulama yazılımlarına da yaptırılabilir.
- Fonksiyon / saklı yordam ile program yazmak, değiştirmek (sürüm kontrolü) ve hata bulmak zordur.

```
#Fonksiyon Örneği
CREATE OR REPLACE FUNCTION inch2m(sayiInch REAL)
RETURNS REAL
AS
$$ -- Fonksiyon govdesinin (tanımının) başlangıcı
BEGIN
    RETURN 2.54 * sayiINCH / 100;
END;
$$ -- Fonksiyon govdesinin (tanımının) sonu
LANGUAGE plpgsql;
```

```
#Fonksiyon çağrısı
SELECT * FROM inch2m(10);

#Koşul İfadeleri
IF miktar < 100 THEN
    ...
ELSEIF miktar >= 100 AND miktar < 200 THEN
    ...
ELSE
    ...
END IF;
#ya da
CASE
    WHEN sonuc > 0 THEN
        ...
    WHEN sonuc < 0 THEN
        ...
    ELSE
        ...
END CASE;
```

Döngüler

LOOP – EXIT/CONTINUE

```
LOOP
    -- some computations
    IF count > 0 THEN
        EXIT;  -- exit loop
    END IF;
END LOOP;

LOOP
    -- some computations
    EXIT WHEN count > 0;  -- same result as previous example
END LOOP;
```

```
LOOP
    -- some computations
    EXIT WHEN count > 100;
    CONTINUE WHEN count < 50;
    -- some computations for count IN [50 .. 100]
END LOOP;
```

FOR

```
FOR i IN 1..10 LOOP
    -- i will take on the values 1,2,3,4,5,6,7,8,9,10 within the loop
END LOOP;

FOR i IN REVERSE 10..1 LOOP
    -- i will take on the values 10,9,8,7,6,5,4,3,2,1 within the loop
END LOOP;

FOR i IN REVERSE 10..1 BY 2 LOOP
    -- i will take on the values 10,8,6,4,2 within the loop
END LOOP;
```

WHILE

```
WHILE amount_owed > 0 AND gift_certificate_balance > 0 LOOP
    -- some computations here
END LOOP;

WHILE NOT done LOOP
    -- some computations here
END LOOP;
```

```
#fonksiyon örneği
CREATE OR REPLACE FUNCTION "fonksiyonTanimlama"(mesaj text, a
RETURNS TEXT -- SETOF TEXT, SETOF RECORD diyerek çok sayıda d
AS
$$
DECLARE
    sonuc TEXT; -- Değişken tanımlama bloğu
BEGIN
    sonuc := '';
    IF tekrarSayisi > 0 THEN
        FOR i IN 1 .. tekrarSayisi LOOP
            sonuc := sonuc || i || '.' || SUBSTRING(mesaj FRO
            -- E: string içerisindeki (E)scape karakterleri i
        END LOOP;
    END IF;
    RETURN sonuc;
END;
$$
LANGUAGE 'plpgsql' IMMUTABLE SECURITY DEFINER;
```

- **IMMUTABLE**: Aynı girişler için aynı çıkışları üretir. Böylece, fonksiyonun gövde kısmı bir kez çalıştırıldıktan sonra diğer çağrılarda çalıştırılmaz. Optimizasyon mümkün olabilir.
- Varsayılan **VOLATILE**: Fonksiyon değeri değişebilir dolayısıyla optimizasyon yapılamaz.
- **SECURITY DEFINER**: Fonksiyon, oluşturan kullanıcının yetkileriyle çalıştırılır.
- Varsayılan **SECURITY INVOKER**: Fonksiyon, çağıran kullanıcının yetkileri ile çalıştırılır.

```
#fonksiyon örneği
CREATE OR REPLACE FUNCTION kayitDolanimi()
RETURNS TEXT
AS
$$
DECLARE
    musteri customer%ROWTYPE; -- customer."CustomerID"%TYPE
    sonuc TEXT;
BEGIN
    sonuc := '';
    FOR musteri IN SELECT * FROM customer LOOP
        sonuc := sonuc || musteri."customer_id" || E'\t' || m
    END LOOP;
    RETURN sonuc;
END;
$$
LANGUAGE 'plpgsql';
customer tablosundaki her müşterinin CustomerID ve
first_name bilgilerini birleştirip, satır satır bir metin diz
olarak döndürür. Fonksiyon, tablodaki tüm kayıtları döngüyle
dolaşarak bir sonuç metni oluşturur ve bu metni geri döner.
#Fonksiyon çağrısı
SELECT  kayitDolanimi();
```

```
#tablo döndürür
CREATE OR REPLACE FUNCTION personelAra(personelNo INT)
RETURNS TABLE(numara INT, adi VARCHAR(40), soyadi VARCHAR(40)
```

```
AS
$$
BEGIN
    RETURN QUERY SELECT "staff_id", "first_name", "last_name"
                 WHERE "staff_id" = personelNo;
END;
$$
LANGUAGE "plpgsql";
#Fonksiyon çağrısı
SELECT * FROM personelAra(1);
```

```
Argüman listesinde çıkış parametresi tanımlanan fonksiyon örn
CREATE OR REPLACE FUNCTION inch2cm(sayiInch REAL, OUT sayiCM
```

Saklı Yordam (Stored Procedure)

- **SELECT, INSERT, UPDATE, DELETE gibi DML komutlarının içerisinden çağrılamaz.**
- **return ile geriye değer döndürmezler.**
- **"call" ifadesi ile çağırılırlar.**
- **"commit"/"rollback" yapılabilir (fonksiyon atomiktir- başlama ve bitiş tek işlemdir. İçerisinde "commit" işlemi yapılamaz).**

```
CREATE OR REPLACE PROCEDURE public.musteriodemelerinihesapla1
-- Mağazaya göre müşteri ödemelerini hesaplayan saklı yordam
    LANGUAGE plpgsql
AS
$$
DECLARE
    customerrow customer%ROWTYPE;
    yoneticiid smallint;
    yonetici record;
    odemetoplami double precision;
BEGIN
    yoneticiid:= (select manager_staff_id from public.store w
    yonetici:= personelAra(yoneticiid);
    FOR customerrow IN SELECT * FROM customer where store_id=
```

```
        LOOP
            odemetoplami:=(SELECT SUM(amount) FROM payment WH
            INSERT INTO musteriodemeleri(id, musteriadi,muste
            VALUES (NEXTVAL('public.sequence1'), customerrow.
        END LOOP;
END;
$$
#çağrı
call musteriodemelerinihesapla1(1::smallint);
```

Özellik	Saklı Yordam (Stored Procedure)	Fonksiyon (Function)
Amaç	Birden fazla işlem gerçekleştirebilir.	Tek bir işlem veya hesaplama döndürür.
Geri Dönen Değer	Genellikle veri döndürmez, işlem yürütür.	Belirli bir değer veya değerler döndürür.
Kullanım Yeri	Veri işleme, güncelleme gibi operasyonel işler.	Hesaplama veya dönüşüm işleri.
Parametreler	Giriş ve çıkış parametreleri alabilir.	Sadece giriş parametreleri alır.
Çağrı	`CALL` komutu ile çağrılır.	`SELECT` veya başka bir sorgu içinde çağrılır.
Veri Manipülasyonu	DML (Data Manipulation Language) işlemleri yapabilir.	DML işlemleri yapabilir, ancak sınırlıdır.
Döngüler ve Koşullar	Daha kompleks işlemleri içerir.	Daha az karmaşık işlemler içerir.
Yan Etkiler	Veritabanı üzerinde yan etkiler (değişiklikler) yaratabilir.	Genellikle yan etkisizdir, sadece döner.
Kullanım	İş akışlarını yönetmek için kullanılır.	Belirli bir sonucu hesaplamak veya döndürmek için kullanılır.
Çağırma Şekli	Bağımsız olarak çağrılabilir.	Sorgular içinde çağrılır.
Hata Yönetimi	Gelişmiş hata yönetimi özelliklerine sahiptir.	Sınırlı hata yönetimi sağlar.

İmleç (Cursor)

-
- İmleç (cursor), sorgu sonucunun toplu olarak oluşturulması yerine parça parça (satır satır) oluşturulmasını sağlar.
- LIMIT ve OFFSET yapılarının da benzer bir işi yaptığını hatırlayınız.
- Yük dengeleme, uygulama sunucusunun, veritabanı sunucusunun ve istemci belleğinin verimli kullanımı vb. amaçlar için kullanılabilir.

```
CREATE OR REPLACE FUNCTION filmAra(yapimYili INTEGER, filmAdi
RETURNS TEXT
AS
$$
DECLARE
    filmAdlari TEXT DEFAULT '';
    film RECORD;
    filmImleci CURSOR(yapimYili INTEGER) FOR SELECT * FROM fi
BEGIN
   OPEN filmImleci(yapimYili);
   LOOP
      FETCH filmImleci INTO film;
      EXIT WHEN NOT FOUND;
      IF film.title LIKE filmAdi || '%' THEN
          filmAdlari := filmAdlari || film.title || ':' || fi
      END IF;
   END LOOP;
   CLOSE filmImleci; #cursor kapatıldı

   RETURN filmAdlari;
END;
$$
LANGUAGE 'plpgsql';
#fonksiyon çağrısı
SELECT * FROM filmAra(2006, 'T');

#filmImleci CURSOR(yapimYili INTEGER) FOR SELECT * FROM film
#WHERE release_year = yapimYili;
#filmImleci adlı bir imleç (cursor) tanımlanır. Bu imleç,
#verilen yapimYili'na göre film tablosundaki tüm filmleri seç
```

Tetikleyici (Trigger)

When	Event	Row-level	Statement-level
BEFORE	INSERT/UPDATE/DELETE	Tables	Tables and views
	TRUNCATE	—	Tables
AFTER	INSERT/UPDATE/DELETE	Tables	Tables and views
	TRUNCATE	—	Tables
INSTEAD OF	INSERT/UPDATE/DELETE	Views	—
	TRUNCATE	—	—

• INSERT, UPDATE ve DELETE (PostgreSQL'de TRUNCATE için de tanımlanabilir) işlemleri ile birlikte otomatik olarak çalıştırılabilen fonksiyonlardır.

```
Ürünlerin birim fiyat değişimlerini izlemek için kullanılan b
tetikleyici örneği aşağıdadır.
CREATE OR REPLACE FUNCTION "urunDegisikligiTR1"()
RETURNS TRIGGER
AS
$$
BEGIN
    IF NEW."UnitPrice" <> OLD."UnitPrice" THEN
        INSERT INTO "UrunDegisikligiIzle"("urunNo", "eskiBiri
        VALUES(OLD."ProductID", OLD."UnitPrice", NEW."UnitPri
    END IF;

    RETURN NEW;
END;
$$
LANGUAGE "plpgsql";
CREATE TRIGGER "urunBirimFiyatDegistiginde"
BEFORE UPDATE ON "products"
FOR EACH ROW
EXECUTE PROCEDURE "urunDegisikligiTR1"();
```

```
Before İfadesi
Ekleme ve güncelleme işleminde yeni verinin
```

```
değiştirilebilmesini/denetimini sağlar
CREATE OR REPLACE FUNCTION "kayitEkleTR1"()
RETURNS TRIGGER
AS
$$
BEGIN
    NEW."CompanyName" = UPPER(NEW."CompanyName"); -- büyük ha
    NEW."ContactName" = TRIM(NEW."ContactName"); -- Önceki ve
    IF NEW."City" IS NULL THEN
            RAISE EXCEPTION 'Sehir alanı boş olamaz';
    END IF;
    RETURN NEW;
END;
$$
LANGUAGE "plpgsql";
CREATE TRIGGER "kayitKontrol"
BEFORE INSERT OR UPDATE ON "customers"  -- veriyi eklemeden/d
FOR EACH ROW
EXECUTE PROCEDURE "kayitEkleTR1"();
```

PostgreSQL Hazır Fonksiyonları

- O anki tarihi seç.

`SELECT CURRENT_DATE;`

LOCALTIME

- O anki zamanı seç.
- Zaman bölgesi olmadan.

CURRENT_TIMESTAMP=NOW()

- O anki tarih ve zamanı birlikte seç.
- TIMESTAMP: Tarih + Zaman
- Zaman bölgesi ile birlikte.

CURRENT_TIME

- O anki zamanı seç.
- Zaman bölgesiyle birlikte.

`SELECT CURRENT_TIME; -- 10:36:58.477505+03`

DATE_PART() / EXTRACT()

- DATE_PART() ve EXTRACT() fonksiyonları, tarih/zaman'dan ya da zaman diliminden(interval) istenen bölümü almak için kullanılır.

DATE_TRUNC

- LOCALTIMESTAMP zaman bölgesi yok

JUSTIFY_DAYS

- Zaman aralığını 30 günlük periyotlara bölerek ifade et.

```
SELECT JUSTIFY_DAYS(interval '51 days');
-- 1 ay 21 gün
```

JUSTIFY_INTERVAL=JUSTIFYDAY+GOURS

TO_CHAR

SELECT TO_CHAR(current_timestamp, 'DD/MM/YYYY'); -- YYYY year (4 basamak), YY, TZ time zone

- Tarih-zaman bilgisini istenilen hassasiyette göstermek için kullanılır.

```
SELECT DATE_TRUNC('minute', timestamp '2018-10-07 23:05:40');
```

EXTRACT EPOCH

- UNIX zaman damgasının başından (1.1.1970'den) belli bri ana kadar geçen süre (sn. cinsinden).

Hafta 11

İleri SQL

Alt Sorgu

WHERE ile Alt Sorgu (Tek Değer Döndüren) Kullanımı:

- **Alt sorgu**, **WHERE** ifadesinde yalnızca **=**, **!=**, **<**, **>**, **<=**, **>=** gibi karşılaştırma operatörleriyle kullanılırsa, alt sorgudan **tek bir değer** döndürmelidir. Aksi halde hata alınır.
- Alt sorgu **AVG**, **COUNT**, **SUM** gibi **çoklu satır fonksiyonları** kullanarak **tek bir değer** döndürebilir.
- **Birincil anahtar** kullanılarak alt sorgudan **tek bir değer** döndürülmesi garanti edilebilir, çünkü birincil anahtar her zaman benzersizdir.

Örnek:

```
SELECT "ProductID", "UnitPrice"
FROM "products"
```

```
WHERE "UnitPrice" < (SELECT AVG("UnitPrice") FROM "product
s");
```

Bu örnekte, alt sorgu **AVG("UnitPrice")**, tüm ürünlerin ortalama fiyatını hesaplar ve tek bir değer döndürür, ardından ana sorgu bu değeri kullanarak **UnitPrice**'ı ortalamanın altında olan ürünleri seçer.

WHERE ile Alt Sorgu (Çok Değer Döndüren) Kullanımı

- Alt sorgudan çok değer dönmesi durumunda IN, ANY ve ALL ifadeleri kullanılmalıdır.

1. **İlk Sorgu:**

   ```
   SELECT "SupplierID"
   FROM "products"
   WHERE "UnitPrice" > 18;
   Bu sorgu, products tablosundaki ürünlerin UnitPrice'ı
   18'den büyük olanların SupplierID'lerini döndürüyor.
   ```

2. **İkinci Sorgu:**

   ```
   SELECT *
   FROM "suppliers"
   WHERE "SupplierID" IN
     (SELECT "SupplierID"
      FROM "products"
      WHERE "UnitPrice" > 18);
   ```

   Bu sorgu, **suppliers** tablosundaki tüm satırları döndürür ancak yalnızca **SupplierID**'si, bir önceki sorgudan dönen **SupplierID**'lerle eşleşen satırları getirir. Burada, alt sorgudan dönen **SupplierID**'leri bir küme olarak kabul edilir ve **suppliers** tablosundaki **SupplierID**'lerin bu kümede olup olmadığına bakılır.

IN Kullanımı:

- **IN** ifadesi, çoklu değerlerle kıyaslama yapmayı sağlar ve özellikle birden fazla değeri **OR** yerine kullanmak için faydalıdır. Bu sorguda, **UnitPrice**'ı

18'den büyük olan ürünlerin **SupplierID**'lerine sahip olan tedarikçilerin bilgilerini almak için kullanılır.

ANY ile Alt Sorgu Kullanımı

- = ANY ifadesi, sorgulanan değerin, alt sorgudan dönen değerler kümesinin elemanlarından her hangi birisine eşit olup olmadığını
- > ANY ifadesi, sorgulanan değerin, alt sorgudan dönen değerler kümesinin elemanlarının her hangi birisinden büyük olup olmadığını
- < ANY ifadesi, sorgulanan değerin, alt sorgudan dönen değerler kümesinin elemanlarının her hangi birisinden küçük olup olmadığını

```
SELECT * FROM "products"
WHERE "UnitPrice" < ANY
(
    SELECT "UnitPrice"
    FROM "suppliers"
    INNER JOIN "products"
    ON "suppliers"."SupplierID" = "products"."SupplierID"
    WHERE "suppliers"."CompanyName" = 'Tokyo Traders'
);
#küçük değer olup olmadığını kotnrol eder
```

ALL ile Alt Sorgu Kullanımı

- > ALL ifadesi, sorgulanan değerin, alt sorgudan dönen değerler kümesinin elemanlarının tamamından büyük olup olmadığını araştırmak için kullanılır.
- < ALL ifadesi, sorgulanan değerin, alt sorgudan dönen değerler kümesinin elemanlarının tamamından küçük olup olmadığını araştırmak için kullanılır.

```
SELECT * FROM "products"
WHERE "UnitPrice" < ALL
(
    SELECT "UnitPrice"
    FROM "suppliers"
    INNER JOIN "products"
    ON "suppliers"."SupplierID" = "products"."SupplierID"
    WHERE "suppliers"."CompanyName" = 'Tokyo Traders'
```

```
);
#tamamınının küçük olup olmadığını kotnrol eder
```

HAVING ile Alt Sorgu Kullanımı

```
SELECT "SupplierID", SUM("UnitsInStock") AS "stoktakiToplamUr
FROM  "products"
GROUP BY "SupplierID"
HAVING SUM("UnitsInStock") < (SELECT AVG("UnitsInStock") FROM
```

Satır İçi (Inline) Alt Sorgu Kullanımı

- Alt sorgular sonucunda tek alan ve tek satır dönmeli. Aksi halde hata verir.

```
SELECT
  "ProductName",
  "UnitsInStock",
  (SELECT MAX("UnitsInStock") FROM "products") AS "enBuyukDeg
FROM "products";
```

İlintili (Correlated) Sorgu

- İç içe döngülerdeki gibi dış sorgunun her bir satırı iç sorguya gönderilerek iç sorgunun çalıştırılması sağlanır.

1. Dış sorgunun birinci satırı seçilir.
2. İç sorgu çalıştırılır ve dış sorguda seçilen satırın SupplierID değerine sahip olan tüm kayıtların UnitPrice alanlarındaki değerlerin aritmetik ortalaması hesaplanır.
3. Dış sorgunun birinci satırının UnitPrice alanındaki değer, alt sorguda hesaplanan ortalamadan büyük ise seçilen birinci satır sonuç kümesine eklenir. Değilse eklenmez.
4. Dış sorgunun ikinci satırı seçilir ve aynı işlem yapılır.
5. Bu işlemler dış sorgunun tüm satırları için tekrarlanır.

```
SELECT "ProductName", "UnitPrice" FROM "products" AS "urunler1"WHERE
"urunler1"."UnitPrice" >
(
  SELECT AVG("UnitPrice") FROM "products" AS "urunler2"WHERE "urunler1"."SupplierID" =
```

```
"urunler2"."SupplierID"
);
```

Dış sorgu: Bu sorgu, **products** tablosunda her bir ürünün **ProductName** (Ürün Adı) ve **UnitPrice** (Birim Fiyat) değerlerini döndürmek için kullanılır.

```
SELECT AVG("UnitPrice")
FROM "products" AS "urunler2"
WHERE "urunler1"."SupplierID" =
"urunler2"."SupplierID"
```

dış sorguda bulunan her bir satır için çalıştırılır. İç sorgu, dış sorgudaki her bir ürünün **SupplierID**'sine göre, aynı tedarikçiye ait olan diğer ürünlerin **UnitPrice** değerlerinin ortalamasını hesaplar. **AVG("UnitPrice")** fonksiyonu, her tedarikçi için ürünlerin ortalama fiyatını döndürür.

💡

ProductID	ProductName	SupplierID	UnitPrice
1	Product A	1	10
2	Product B	1	20
3	Product C	2	15
4	Product D	2	12
5	Product E	3	25
6	Product F	3	30

- İlk Satır: `urunler1` (Product A - SupplierID = 1)
- **Dış sorgu**: İlk satır **Product A** (SupplierID = 1, UnitPrice = 10) için çalışır.
- **İç sorgu**: `SupplierID = 1` olan ürünlerin **UnitPrice** ortalaması hesaplanır:
  - **Product A**: 10
  - **Product B**: 20
  - Ortalama = `(10 + 20) / 2 = 15`
- **Dış sorgu koşulu**: `10 > 15` (Product A'nın fiyatı, ortalama fiyattan büyük mü?) — **Hayır**, bu nedenle **Product A** sonuçlara eklenmez.

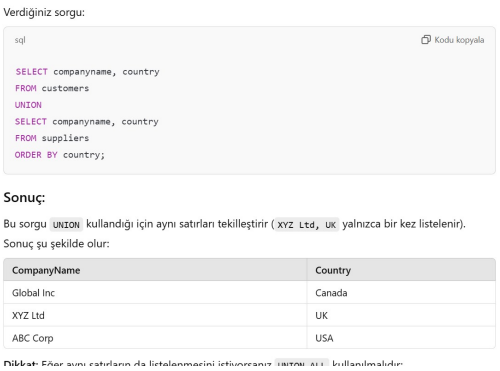
İkinci Satır: `urunler1` (Product B - SupplierID = 1)

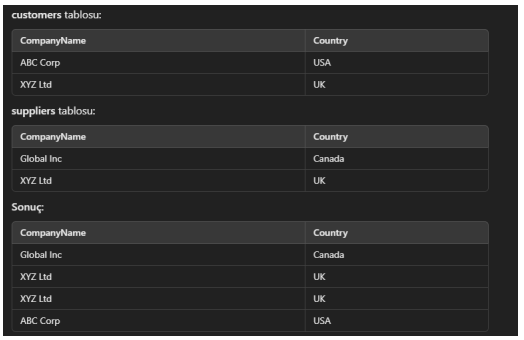
- **Dış sorgu**: İkinci satır **Product B** (SupplierID = 1, UnitPrice = 20) için çalışır.
- **İç sorgu**: `SupplierID = 1` olan ürünlerin **UnitPrice** ortalaması zaten hesaplanmıştı (15).
- **Dış sorgu koşulu**: `20 > 15` (Product B'nin fiyatı, ortalama fiyattan büyük mü?) — **Evet**, bu nedenle **Product B** sonuçlara eklenir. böyle devam eder...

UNION ve UNION ALL

- İki tablonun küme birleşimini alır.
- Rastgele iki tablonun birleşimi alınamaz.

    - İki tablonun nitelik sayıları aynı olmalı.
    - Aynı sıradaki nitelikleri aynı değer alanı üzerinde tanımlanmış olmalıdır.
- UNION ifadesi ile aynı kayıtlar bir defa gösterilir.
- UNION ALL ifadesi ile aynı kayıtlar gösterilir.






```
SELECT "CompanyName", "Count
UNION
SELECT "CompanyName", "Count
ORDER BY 2;
alfabetik sıra
```

```
SELECT "CompanyName", "Count
UNION ALL
SELECT "CompanyName", "Count
ORDER BY 2;
```

INTERSECT

- İki tablonun küme kesişimi elde edilir.
- Rasgele iki tablonun kesişimi alınamaz.
    - İki tablonun nitelik sayıları aynı olmalı.
    - Aynı sıradaki nitelikleri aynı değer alanı üzerinde tanımlanmış olmalı.

`INTERSECT` işlemi, iki sorgudan sadece **ortak** olan (yani her iki tabloda da bulunan) satırları döndürür.

EXCEPT

- Bir tablonun diğerinden farkını elde etmek için kullanılır.
- Rastgele iki tabloya uygulanamaz.
    - İki tablonun nitelik sayıları aynı olmalı.

- Aynı sıradaki nitelikleri aynı değer alanı üzerinde tanımlanmış olmalı.

Hareket/İşlem (Transaction)

- ACID(Atomicity, Consistency, Isolation,Durability)belirtilen ozellikdestekler.
- Atomicity (Atomiklik): Hareket/işlem (transaction) kapsamındaki alt işlemlerin tamamı bir bütün olarak ele alınır. Ya alt işlemlerin tamamı başarılı olarak çalıştırılır, ya da herhangi birinde hata varsa tamamı iptal edilir ve veritabanı eski kararlı haline döndürülür.

> 💡 Alice, Bob'a 100 TL göndermek istiyor.1)Alice'in hesabından 100 TL çekilmesi2)Bob'un hesabına 100 TL eklenmesi.Eğer ilk adım başarılı olur, ancak ikinci adım başarısız olursa, **Atomicity** devreye girer. Tüm işlem geri alınır ve Alice'in hesabından para çekilmesi iptal edilir. Bu, veritabanının tutarlılığını sağlar.

- Consistency (Tutarlılık): Herhangi bir kısıt ihlal edilirse roll back işlemiyle veritabanı eski kararlı haline döndürülür.mesela bakiye negatif olamaz diye bir ksııt varsa fazla para çekilemez.
- Isolation (Yalıtım): İşlemler birbirlerini (ortak kaynak kullanımı durumunda) etkilemezler. Kullanılan ortak kaynak işlem tarafından, işlem tamamlanana kadar, kilitlenir.mesela 2 kişi para çekmek istiyor bu birbirini etkileyen bir durum olmamalı
- Durability (Mukavemet): Sistem tarafından bir hata meydana gelmesi durumunda tamamlanmamış olan işlem sistem çalışmaya başladıktan sonra mutlaka tamamlanır.

Hafta 12.1

Normalizasyon

- Normalizasyon, veri fazlalıklarını en aza indirerek veri düzensizliklerinin (data anomaly) önüne geçebilmek için tablo yapılarını değerlendirme ve düzeltme işlemi olarak tanımlanabilir.

- Normalizasyon işlemi normal form adı verilen seri işlemlerden meydana gelir. 1NF, 2NF, 3NF, 4NF
- Her tasarım için en yüksek NF daha iyi sonuç verir denemez. Yüksek başarıma ihtiyaç duyulan bazı durumlarda normal formun (NF) düşürülmesi (denormalizasyon) gerekebilir.

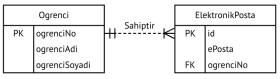
Birinci Normal Form (1NF)

- Tüm alanlar birincil anahtar tarafından belirlenebilmelidir.
- Tüm alanlar tek değerli olmalıdır.

ogrenciNo	ogrenciAdi	ogrenciSoyadi	ePosta
B05051005	Merve	Şahin	msahin@sakarya.edu.tr, msahin@gmail.com
B01031003	Mehmet	Arslan	mehmet.arslan@sakarya.edu.tr
B01032001	Hakan	Demir	hdemir@sakarya.edu.tr
B03013001	Filiz	Şahin	f.sahin@sakarya.edu.tr

ogrenciNo	ogrenciAdi	ogrenciSoyadi
B05051005	Merve	Şahin
B01031003	Mehmet	Arslan
B01032001	Hakan	Demir
B03013001	Filiz	Şahin

id	ePosta	ogrenciNo
1	msahin@sakarya.edu.tr	B05051005
2	msahin@gmail.com	B05051005
3	mehmet.arslan@sakarya.edu.tr	B01031003
4	hdemir@sakarya.edu.tr	B01032001
5	f.sahin@sakarya.edu.tr	B03013001



Normalizasyon sonucunda

Tekrarlanan Veri Grupları

- Tabloda birincil anahtar var.
- Her sütunda tek değer var.
- Buna rağmen veri tekrarı vardır.

ogrenciNo	dersNo	dersAdi	kredi	not
B05051005	T001	Türkçe	4	80
B01031003	T002	Tarih	3	70
B01032001	C001	Coğrafya	3	75
B03013001	T001	Türkçe	4	85
B01013003	C001	Coğrafya	3	97

İkinci Normal Form (2NF)

- Anahtarlar belirlenirken fonksiyonel bağımlılık göz önüne alınmalıdır.
- Aşağıdaki tabloda, ogrenciNo niteliği kullanılarak adi alanı belirlenebilir. (**tersi doğru değildir**)
- Bu durumda:
  - **ogrenciNo** alanı **adi** alanını **belirler**.
  - **adi** alanı, **ogrenciNo** alanına **fonksiyonel bağımlıdır (ogrenciNo → ogrenciAdi)**.

Ogrenciler

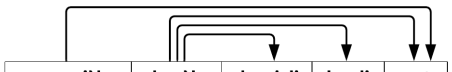
ogrenciNo TC Kimlik No	adi	soyadi	sifre md5 formatında saklanıyor...	telefonNo	eposta	babaAdi	adres	dogumTarihi	il	ilce
00000000001	Ayşe	Demirr	a7f4e18520f1a28fb9b1edb53f9fd6b6		ad@a.com	Hasan	Bilinmiyor	0000-00-00	34	409
00000000003	Hasan	Çelik	hasancelik		hc@a.com	Hasan	Bilinmiyor	*NULL*	01	001
00000000004	Tamer	Yorulmaz	e1e6205a7c630320a8f854df101905fb		ty@a.comm	Yılmaz	Bilinmiyor	1975-05-01	01	008
00000000008	Ayşe	Eren	e78c265a4f809993ccb24c6ea5c308dc		aer@a.com	Mustafa	Konya	1994-06-07	42	560
00000000009	Ayşe	Yılmaz	9693bb4495eae586d84e2001f1d665ac		ay@a.com	Ahmet	Kocaeli	1999-05-01	41	533
00000000021	Ayten	Gül	035e15c85c630a56ebfd9d44f7796da1	1234567892	Girilmemiş	Girilmemiş	Gebze	1993-09-01	00	940

Tam Fonksiyonel Bağımlılık

- Nitelikler birden fazla alanın birleşimine fonksiyonel bağımlı olabilir.
  - ogrenciNo, dersNo ➔ ortalama **Tam fonksiyonel bağımlılık**
  - ogrenciNo, dersNo ➔ dersAdi **Kısmi fonksiyonel bağımlılık**
  - "dersAdi" sadece "dersNo"ya bağlıdır, çünkü bir dersin adı tüm öğrencilere aynı olabilir. "dersAdi", "ogrenciNo" ile **bağımlı değildir** ve **kısmi bağımlılık** oluşturur. Öğrencinin dersin adını öğrenmesi için yalnızca "dersNo" yeterlidir. "ortalama" değeri, yalnızca "ogrenciNo" ve "dersNo" birleşimine tamamen bağlıdır. Yani, her öğrenci ve ders için bir ortalama hesaplanır

İkinci Normal Form Şartları

- Tablonun birinci normal formda olması gerekir.
- Birincil anahtar, birden fazla alanın birleşiminden oluşuyorsa, tablonun 2NF'de olabilmesi için diğer alanların birincil anahtara **tam fonksiyonel bağımlı** olması gerekir.
- Birincil anahtar tek alandan oluşuyorsa ve tablo 1NF'de ise, 2NF de sağlanmış olur.



ogrenciNo	dersNo	dersAdi	kredi	not
B05051005	T001	Türkçe	4	80
B01031003	T002	Tarih	3	70
B01032001	C001	Coğrafya	3	75
B03013001	T001	Türkçe	4	85
B01013003	C001	Coğrafya	3	97

ogrenciNo	dersNo	not
B05051005	T001	80
B01031003	T002	70
B01032001	C001	75
B03013001	T001	85
B01013003	C001	97

dersNo	dersAdi	kredi
T001	Türkçe	4
T002	Tarih	3
C001	Coğrafya	3

Tabloyu 2NF'ye Dönüştürme (Kısmi Bağımlılıkların Giderilmesi)

Üçüncü Normal Form (3NF)

Geçişken Bağımlılık

- Eğer A ➔ B ve B ➔ C ise A ➔ B ➔ C
  - A, B üzerinden C'yi belirler.
  - C, A ya geçişken bağımlıdır.
- oduncNo ➔ ISBNNo ➔ kitapAdi
- oduncNo ➔ ISBNNo ➔ yayinYili
- oduncNo alanı, ISBNNo alanı üzerinden kitapAdi alanını belirler.
- kitapAdi alanı, oduncNo alanına geçişken bağımlıdır.

oduncNo	uyeNo	kitapNo	kitapAdi	yayinYili	oduncTarihi	teslimTarihi
1	1000	1	Veri Yapıları	2017	2017-07-01	2017-07-10
2	1001	1	Veri Yapıları	2017	2017-05-20	2017-06-01
3	1002	2	İşaretler Sistemler	2005	2017-11-10	2017-11-15
4	1003	3	Sayısal Analiz	2001	2017-03-20	2017-03-27
5	1001	3	Sayısal Analiz	2001	2017-11-18	2017-11-18

Üçüncü Normal Form Şartları

- Tablo 2NF'de ise ve geçişken bağımlılık yok ise 3NF'dedir.

Tabloyu 3NF'ye Dönüştürme

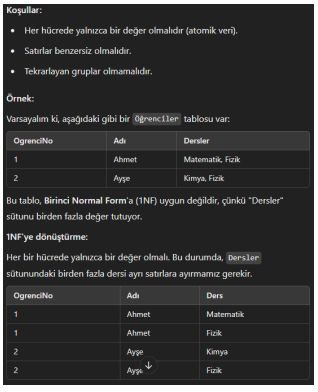
oduncNo	uyeNo	kitapNo	kitapAdi	yayinYili	oduncTarihi	teslimTarihi
1	1000	1	Veri Yapıları	2017	2017-07-01	2017-07-10
2	1001	1	Veri Yapıları	2017	2017-05-20	2017-06-01
3	1002	2	İşaretler Sistemler	2005	2017-11-10	2017-11-15
4	1003	3	Sayısal Analiz	2001	2017-03-20	2017-03-27
5	1001	3	Sayısal Analiz	2001	2017-11-18	2017-11-18

oduncNo	uyeNo	kitapNo	oduncTarihi	teslimTarihi
1	1000	1	2017-07-01	2017-07-10
2	1001	1	2017-05-20	2017-06-01
3	1002	2	2017-11-10	2017-11-15
4	1003	3	2017-03-20	2017-03-27
5	1001	3	2017-11-18	2017-11-18

kitapNo	kitapAdi	yayinYili
1	Veri Yapıları	2017
2	İşaretler Sistemler	2005
3	Sayısal Analiz	2001

Özet

- **1NF: Birincil anahtar mevcuttur ve çok değerli alanlar yoktur.**
- **2NF: Birinci normal formdadır ve kısmi bağımlılık yoktur.**
- **3NF: İkinci normal formdadır ve geçişken bağımlılık yoktur.**



Koşullar:

- Her hücrede yalnızca bir değer olmalıdır (atomik veri).
- Satırlar benzersiz olmalıdır.
- Tekrarlayan gruplar olmamalıdır.

Örnek:

Varsayalım ki, aşağıdaki gibi bir `Öğrenciler` tablosu var:

OgrenciNo	Adı	Dersler
1	Ahmet	Matematik, Fizik
2	Ayşe	Kimya, Fizik

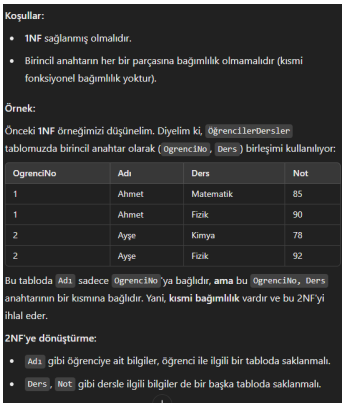
Bu tablo, **Birinci Normal Form**'a (1NF) uygun değildir, çünkü "Dersler" sütunu birden fazla değer tutuyor.

1NF'ye dönüştürme:

Her bir hücrede yalnızca bir değer olmalı. Bu durumda, `Dersler` sütunundaki birden fazla dersi ayrı satırlara ayırmamız gerekir.

OgrenciNo	Adı	Ders
1	Ahmet	Matematik
1	Ahmet	Fizik
2	Ayşe	Kimya
2	Ayşe	Fizik

1NF



Koşullar:

- **1NF** sağlanmış olmalıdır.
- Birincil anahtarın her bir parçasına bağımlılık olmamalıdır (kısmi fonksiyonel bağımlılık yoktur).

Örnek:

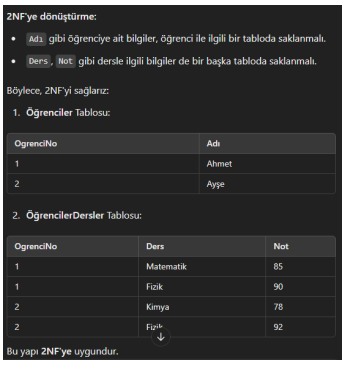
Önceki **1NF** örneğimizi düşünelim. Diyelim ki, `ÖğrencilerDersler` tablomuzda birincil anahtar olarak (`OgrenciNo`, `Ders`) birleşimi kullanılıyor:

OgrenciNo	Adı	Ders	Not
1	Ahmet	Matematik	85
1	Ahmet	Fizik	90
2	Ayşe	Kimya	78
2	Ayşe	Fizik	92

Bu tabloda `Adı` sadece `OgrenciNo`'ya bağlıdır, **ama** bu `OgrenciNo`, `Ders` anahtarının bir kısmına bağlıdır. Yani, **kısmi bağımlılık** vardır ve bu 2NF'yi ihlal eder.

2NF'ye dönüştürme:

- `Adı` gibi öğrenciye ait bilgiler, öğrenci ile ilgili bir tabloda saklanmalı.
- `Ders`, `Not` gibi dersle ilgili bilgiler de bir başka tabloda saklanmalı.



2NF'ye dönüştürme:

- `Adı` gibi öğrenciye ait bilgiler, öğrenci ile ilgili bir tabloda saklanmalı.
- `Ders`, `Not` gibi dersle ilgili bilgiler de bir başka tabloda saklanmalı.

Böylece, 2NF'yi sağlarız:

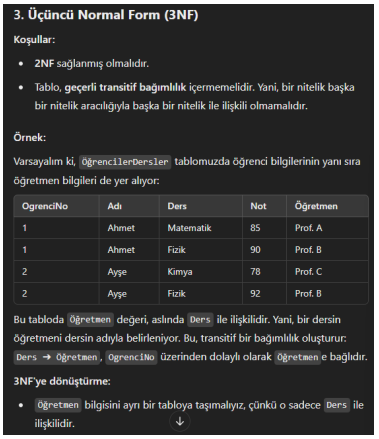
1. **Öğrenciler** Tablosu:

OgrenciNo	Adı
1	Ahmet
2	Ayşe

2. **ÖğrencilerDersler** Tablosu:

OgrenciNo	Ders	Not
1	Matematik	85
1	Fizik	90
2	Kimya	78
2	Fizik	92

Bu yapı **2NF'ye** uygundur.



3. Üçüncü Normal Form (3NF)

Koşullar:

- **2NF** sağlanmış olmalıdır.
- Tablo, **geçerli transitif bağımlılık** içermemelidir. Yani, bir nitelik başka bir nitelik aracılığıyla başka bir nitelik ile ilişkili olmamalıdır.

Örnek:

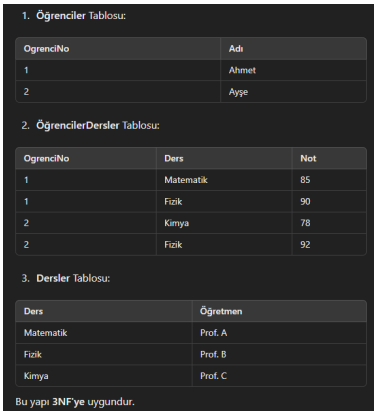
Varsayalım ki, `ÖğrencilerDersler` tablomuzda öğrenci bilgilerinin yanı sıra öğretmen bilgileri de yer alıyor:

OgrenciNo	Adı	Ders	Not	Öğretmen
1	Ahmet	Matematik	85	Prof. A
1	Ahmet	Fizik	90	Prof. B
2	Ayşe	Kimya	78	Prof. C
2	Ayşe	Fizik	92	Prof. B

Bu tabloda `Öğretmen` değeri, aslında `Ders` ile ilişkilidir. Yani, bir dersin öğretmeni dersin adıyla belirleniyor. Bu, transitif bir bağımlılık oluşturur: `Ders` → `Öğretmen`, `OgrenciNo` üzerinden dolaylı olarak `Öğretmen`e bağlıdır.

3NF'ye dönüştürme:

- `Öğretmen` bilgisini ayrı bir tabloya taşımalıyız, çünkü o sadece `Ders` ile ilişkilidir.



1. **Öğrenciler** Tablosu:

OgrenciNo	Adı
1	Ahmet
2	Ayşe

2. **ÖğrencilerDersler** Tablosu:

OgrenciNo	Ders	Not
1	Matematik	85
1	Fizik	90
2	Kimya	78
2	Fizik	92

3. **Dersler** Tablosu:

Ders	Öğretmen
Matematik	Prof. A
Fizik	Prof. B
Kimya	Prof. C

Bu yapı **3NF**'ye uygundur.

HAFTA 12.2

Başarım Eniyileme (Performance Optimization)

EXPLAIN ile pg_statistic katalog tablosuna dayalı olarak, başarımla ilgili tahmini değerler döndürülür (cost=0.00..16.49 rows=1 width=70).

ANLYZE ile birlikte sorgu gerçekten çalıştırılarak gerçek değerler döndürülür (actual time=0.115..0.181 rows=1 loops=1).

1. EXPLAIN ANALYZE Kullanımı

Örnek: Basit SELECT Sorgusu

```
EXPLAIN ANALYSE
SELECT * FROM "customer";
------------------------------
Seq Scan on customer  (cost=0.00..14.99 rows=599 width=70)
(actual time=0.009..0.087 rows=599 loops=1)
Planning Time: 0.092 ms   // uygun planı seçme zamanı
Execution Time: 0.139 ms // Seçilen planın çalıştırılma zam
anı.
//Actual Time içerisine oturum açma/kapatma zamanı eklenere
k bulunur
Seq Scan on customer  (cost=0.00..14.99 rows=599 width=70)
(actual
time=0.009..0.087 rows=599 loops=1)
```

```
Planning Time: 0.092 ms
Execution Time: 0.139 ms
```

- **cost**: İlk ve tüm satırların getirilme maliyeti.
- **rows**: Getirilecek satır sayısı.
- **width**: Satırların ortalama byte boyutu.
- **actual time**: Gerçek zamanlama bilgisi.

Örnek: WHERE Koşulu Kullanımı

```
EXPLAIN ANALYSE
SELECT * FROM "customer"
WHERE "first_name" = 'Bruce';
------------------------------
Seq Scan on customer  (cost=0.00..16.49 rows=1 width=70)
(actual time=0.115..0.181 rows=1 loops=1)
  Filter: ((first_name)::text = 'Bruce'::text)
  Rows Removed by Filter: 598
Planning Time: 0.110 ms
Execution Time: 0.202 ms
```

Bu sorgu, filtreleme nedeniyle çok sayıda satırı atar. `EXPLAIN ANALYZE` ile hangi işlemlerin hangi sürelerde gerçekleştiğini gözlemleyebilirsiniz.

Örnek: Join Kullanımı

```
EXPLAIN ANALYSE
SELECT   "public"."customer"."customer_id",
"public"."customer"."first_name",
"public"."customer"."last_name",
"public"."address"."phone"
FROM     "customer"
INNER JOIN "address"  ON "customer"."address_id" = "addres
s"."address_id"
------------------------------
Hash Join  (cost=21.57..38.14 rows=599 width=29) (actual ti
me=0.510..0.972 rows=599 loops=1)
```

```
Hash Cond: (customer.address_id = address.address_id)
->  Seq Scan on customer  (cost=0.00..14.99 rows=599 width=
19) (actual time=0.009..0.097 rows=599 loops=1)
->  Hash  (cost=14.03..14.03 rows=603 width=16) (actual tim
e=0.485..0.485 rows=603 loops=1)
Buckets: 1024  Batches: 1  Memory Usage: 39kB
->  Seq Scan on address  (cost=0.00..14.03 rows=603 width=1
6) (actual time=0.011..0.237 rows=603 loops=1)
Planning Time: 0.642 ms
Execution Time: 1.052 ms
```

Bu sorguda **Hash Join** yöntemi kullanılmıştır, yani PostgreSQL sıralı olarak `customer` ve `address` tablolarını tarayıp bunları birleştiriyor.

2. Projeksiyon: Yalnızca Gerekli Alanların Seçilmesi

• SELECT ifadesinde bütün alanlara projeksiyon yapmak (* kullanımı) yerine yalnızca gerekli olan alanlara projeksiyon yapmalıyız. Yani yalnızca gerekli alanların getirilmesini istemeliyiz. Böylece, işlem gecikmesi, iletim gecikmesi ve kaynak kullanımı azaltılmış olur.

Örnek: Tüm Alanları Seçmek (İyi Değil)

```
EXPLAIN ANALYZE
SELECT *
FROM "customer"
INNER JOIN "store" ON "customer"."store_id" = "store"."stor
e_id"
INNER JOIN "rental" ON "rental"."customer_id" = "custome
r"."customer_id"
INNER JOIN "inventory" ON "inventory"."store_id" = "stor
e"."store_id"
INNER JOIN "film" ON "inventory"."film_id" = "film"."film_i
d";
Execution time: 10968.823 ms
```

Örnek: Yalnızca Gerekli Alanları Seçmek

```
EXPLAIN ANALYZE
SELECT "customer"."first_name", "customer"."last_name", "fi
lm"."film_id", "film"."title"
FROM "customer"
INNER JOIN "store" ON "customer"."store_id" = "store"."stor
e_id"
INNER JOIN "rental" ON "rental"."customer_id" = "custome
r"."customer_id"
INNER JOIN "inventory" ON "inventory"."store_id" = "stor
e"."store_id"
INNER JOIN "film" ON "inventory"."film_id" = "film"."film_i
d";
Execution time: 6220.990 ms
```

Bu örnekte, sadece gerekli sütunlar seçildiği için önceki sorguya göre daha hızlı bir sonuç elde edilmiştir.

3. LIMIT ve OFFSET Kullanımı

Veri setinin bir kısmını almak, büyük veri setlerinde performansı iyileştirebilir.

Örnek: LIMIT ve OFFSET Uygulaması

```
EXPLAIN ANALYZE
SELECT "store"."store_id", "film"."title"
FROM "inventory"
INNER JOIN "film" ON "inventory"."film_id" = "film"."film_i
d"
INNER JOIN "store" ON "inventory"."store_id" = "store"."sto
re_id"
LIMIT 20 OFFSET 40;
Execution time: 0.315 ms
```

Bu sorgu, yalnızca 40. satırdan sonrasındaki 20 kaydı getirir.offsetten sonraki değerden itibaren

4. Gereksiz Sıralamadan Kaçınmak

Gereksiz sıralama işlemleri, sorgu performansını düşürebilir.

Örnek: Sıralama ile Performans

```
EXPLAIN ANALYZE
SELECT "store"."store_id", "film"."title"
FROM "inventory"
INNER JOIN "film" ON "inventory"."film_id" = "film"."film_i
d"
INNER JOIN "store" ON "inventory"."store_id" = "store"."sto
re_id"
ORDER BY "film"."title";
Execution time: 7.411 ms
```

Sıralama işlemi genellikle gereksiz yere zaman alabilir. Bu nedenle sıralama yalnızca gerçekten gerekli olduğunda kullanılmalıdır.

5. Index Kullanımı

İndeksler, belirli alanlarda yapılan aramaları hızlandırabilir.

Örnek: İndeks Kullanımı ile Performans İyileştirme

Öncelikle, `adi` alanı için bir indeks oluşturulmuştur:

```
>CREATE OR REPLACE FUNCTION "veriGir"(kayitSayisi integer)
RETURNS VOID
AS
$$
BEGIN
    IF kayitSayisi > 0 THEN
        FOR i IN 1 .. kayitSayisi LOOP
            insert into "Kisiler" ("adi","soyadi", "kayitTa
rihi")
            Values(
                substring('ABCÇDEFGĞHIiJKLMNOÖPRSŞTUÜVYZ' f
rom ceil(random()*10)::smallint for ceil(random()*20)::SMAL
LINT),
                substring('ABCÇDEFGĞHIiJKLMNOÖPRSŞTUÜVYZ' f
rom ceil(random()*10)::smallint for ceil(random()*20)::SMAL
LINT),
                NOW() + (random() * (NOW()+'365 days' - NOW
```

```
()))
                 );
        END LOOP;
    END IF;
END;
$$
LANGUAGE 'plpgsql'  SECURITY DEFINER;


>SELECT "veriGir"(100000);


>EXPLAIN ANALYZE
SELECT * FROM "Kisiler"
WHERE "adi"='DENEME' -- Satırlardan birinin adi alanı "DENE
ME" olarak değiştirilmeli
----------------------------------------
Seq Scan on "Kisiler"  (cost=0.00..2107.00 rows=496 width=3
8) (actual time=20.214..20.215 rows=1 loops=1)
  Filter: ((adi)::text = 'DENEME'::text)
  Rows Removed by Filter: 99999
Planning Time: 0.085 ms
Execution Time: 20.237 ms
```

Birleşim (INNER JOIN), IN ve EXIST (İlintili Sorgu)

- İlintili sorgu, özellikle EXIST ifadesi ile birlikte, daha iyi sonuç verebilir.

HAVING

- HAVING ifadesi seçim işlemi yapılıp gruplandırma işlemi tamamlandıktan sonra filtreleme yapmak için kullanılır. Filtreyi, mümkünse gruplama işleminden önce eklemek başarımı artırır.

```
EXPLAIN ANALYSE
SELECT "category"."name", COUNT("film"."film_id")
FROM "film"
LEFT OUTER JOIN "film_category" ON "film"."film_id" = "film_c
LEFT OUTER JOIN "category" ON "film_category"."category_id" =
```

```
GROUP BY "category"."name"
HAVING "category"."name" = 'Horror' OR "category"."name" = 'C
```

Alt Sorgu Sayısı

- Bazen ana sorguda birden fazla alt sorgu bulunabilir. Bu durumda alt sorgu bloklarının sayısını azaltmaya çalışmalıyız.

UNION ve UNION ALL

- UNION yerine UNION ALL komutunu kullanmaya çalışmalıyız. UNION komutu icra edilirken DISTINCT işlemi de gerçekleştirildiği için daha yavaştır.

Genel Kurallar

- Belirli boyutu aşan ikili nesneleri (resim, pdf vb.) depolamak için, ilk olarak onları dosyalama sistemine yerleştiriniz ve sonrasında veritabanına dosyanın konumunu ekleyiniz.

VACUUM ve ANALYSE

- PostgreSQL'de bir kayıt silindiği zaman aslında gerçekten silinmez.
- Yalnızca silindiğine ilişkin bir işaret olur. Dolayısıyla belirli bir süre sonra depolama alanı problemi oluşabilir. Silinen kayıtların gerçekten tablodan silinmesini gerçekleştirmek için VACUUM komutu kullanılır. Bu yapıldığında depolama alanımızda yer açılacaktır.

vacumm full tabloları kilitleyerek yeni bir kopyasını oluşturur ve daha sonra eski tabloyu siler.