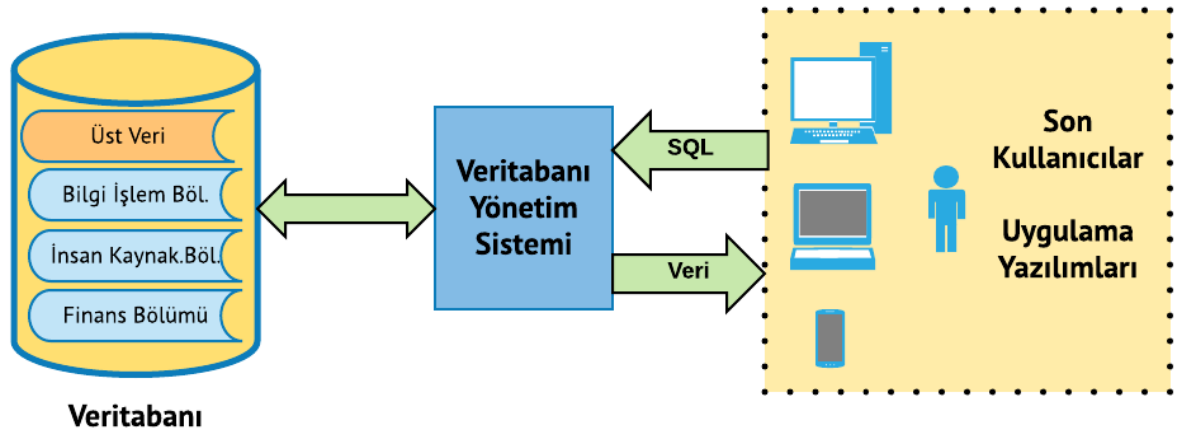


BÖLÜM 1

- İşlenerek anlam kazandırılmamış ham gerçeklere veri denir.
- Veriler işlenerek bilgi oluşturulur.
- **Veri yönetimi** verinin uygun bir şekilde üretimi, saklanması ve erişilmesiyle ilgilenen disiplindir.
- Veriler, kayıtlar halinde klasik dosya yapısı kullanılarak saklanır.



Veri Tümlleştirme (Data Integration): Verilerin tekrarsız olarak etkin bir şekilde saklanması garanti edilebilir.

Veri Bütünlüğü (Data Integrity): Verilerin bozulmadan ve tutarlı olarak saklanması sağlanabilir. Kısıtlar eklenerek veri tutarsızlığı önlenir (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.).

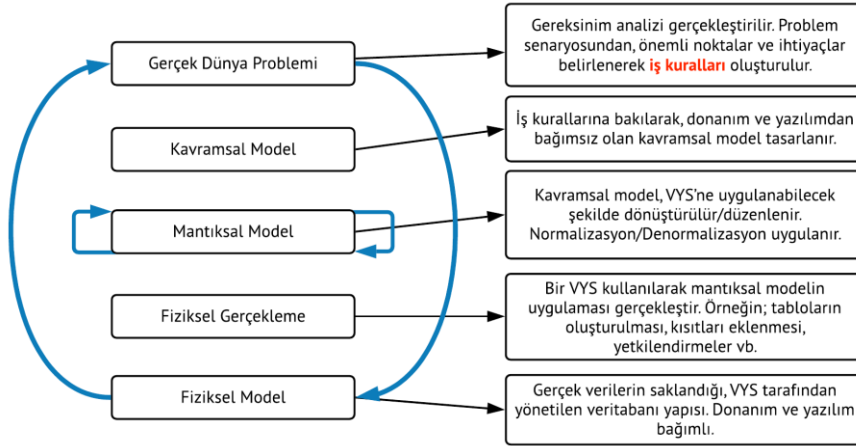
Veri Soyutlama (Data Abstraction): Kullanıcıya, karmaşık yapıdaki fiziksel veri yapısı yerine anlaşılabilirliği ve yönetilebilirliği daha kolay olan mantıksal model sunulur.

Veritabanı Sınıfları Kullanım Amacı

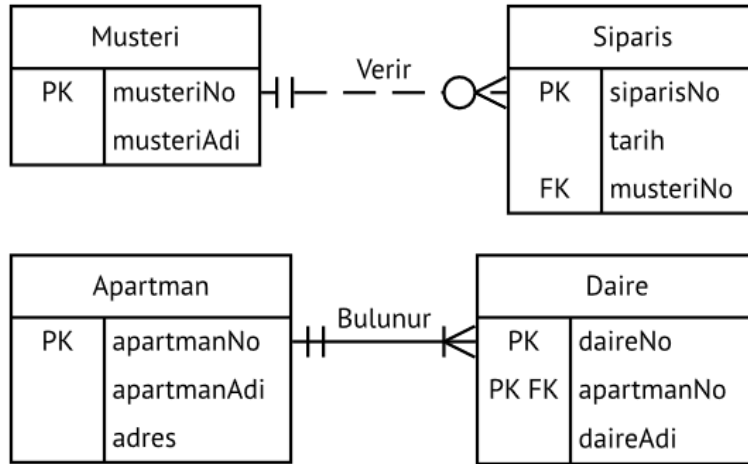
Operasyonel: Veriler üzerinde sürekli değişiklikler yapılır. (OLTP: Online Transaction Processing)

Veri Ambarı: Veriler raporlama ve karar destek amaçlarıyla kullanılır. (OLAP: Online Analytical Processing)

BÖLÜM 2



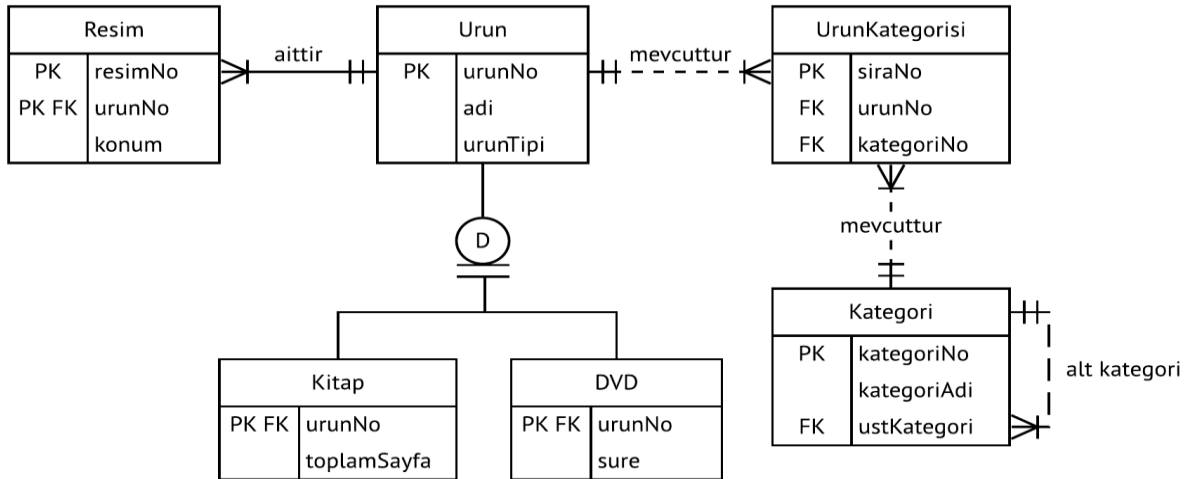
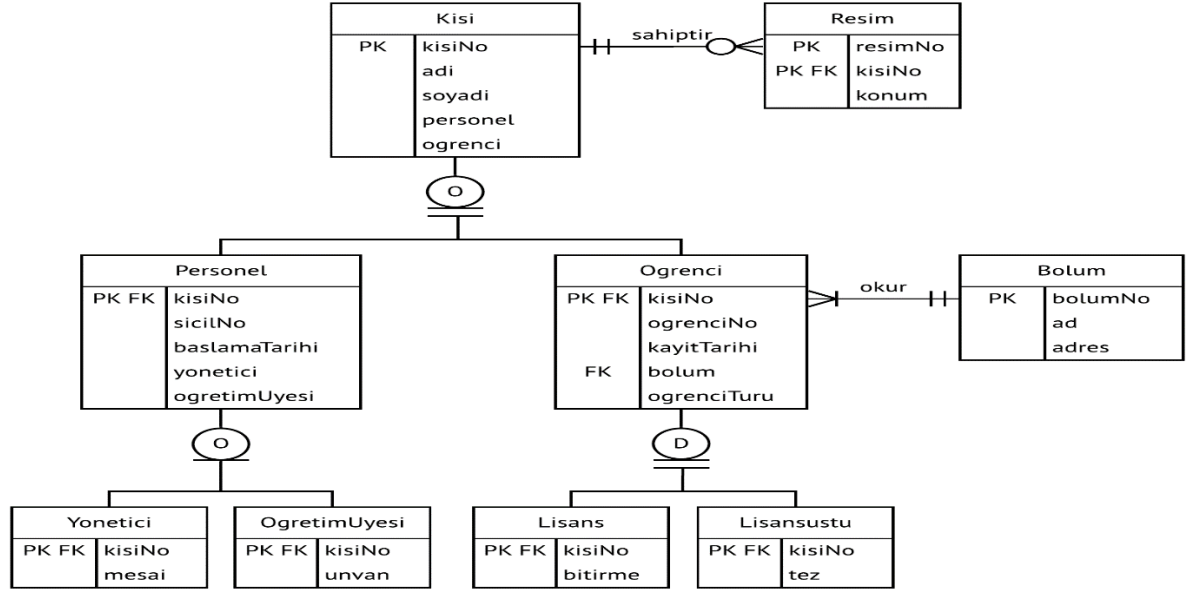
- Kesikli ise zayıf bağlantı , Düz çizgi ise güçlü bağlantı vardır.



BÖLÜM 3

DİSJOİNT (D)	OVERLEAP (O)
Aynı temel varlıkla ilgili tek çocuk olabilir	Aynı temel varlıkla ilgili birçok çocuk olabilir

Tek Çizgi	Çift Çizgi
Üst tip alt tip olmadan da olabilir	Üst tip kaydın kesinlikle alt tip kaydı olması lazım



BÖLÜM 4

DISTINCT	Tekillliği sağlar	<code>SELECT DISTINCT "City" from "customers";</code>
ORDER BY	Sıralama yapar	<code>SELECT * FROM "customers" ORDER BY "ContactName" ASC;</code> ASC -> ARTAN DESC -> azalan
LIKE	Belirli paterne bakar	<code>SELECT * FROM "customers" WHERE "Country" LIKE '%pa%';</code> %-> birden fazla _-> sadece 1 tane
BETWEEN	Aralık kontrol eder	<code>SELECT * FROM "products" WHERE "ProductName" BETWEEN 'C' AND 'M';</code>
IN	İçinde olup olmadığına bakar	<code>SELECT * FROM "customers" WHERE "public"."customers"."Country" IN ('Türkiye', 'Kuzey Kıbrıs Türk Cumhuriyeti');</code>
AS	Takma isim verir	<code>SELECT "CompanyName" AS "musteriler" FROM "customers";</code>
SELECT ... INTO	Bir tabloyu başka tabloya kopyalamaya yarar	<code>SELECT * INTO "MusteriYedek" FROM "customers";</code>
INSERT	Ekleme işi yapar	<code>INSERT INTO "MusteriYedek" SELECT "CompanyName", "ContactName" FROM "customers";</code>
UPDATE	Güncelleme Yapar	<code>UPDATE "customers" SET "ContactName" = 'Mario Pontes', "City" = 'Rio de Janeiro' WHERE "CompanyName" = 'Familia Arquibaldo';</code>
DELETE	Silme işi Yapar	<code>DELETE FROM "customers" WHERE "CompanyName" = 'LINO-Delicateses' AND "ContactName" = 'Felipe Izquierdo';</code>
COUNT	Sayma işi yapar	<code>SELECT COUNT("Region") FROM "customers" WHERE "Country" = 'Mexico';</code>
LIMIT	Limit belirtiriz	<code>SELECT * FROM "products" ORDER BY "ProductID" ASC LIMIT 4;</code>
MAX / MIN	Seçilen sütundaki en büyük/küçük değere ulaşmak için kullanılır	<code>SELECT MAX("UnitPrice") FROM "products";</code>
SUM	Toplam için kullanılır	<code>SELECT SUM("UnitPrice") AS "toplam" FROM "products";</code>
AVG	Ortalama almak için kullanılır	<code>SELECT AVG("UnitPrice") FROM "products";</code>
GROUP BY	Gruplamak için kullanılır	<code>SELECT "SupplierID", SUM("UnitsInStock") AS "stokSayisi" FROM "products" GROUP BY "SupplierID";</code>
HAVING	Gruplanmışları filtreler	<code>SELECT "SupplierID", COUNT("SupplierID") AS "urunSayisi" FROM "products" GROUP BY "SupplierID" HAVING COUNT("SupplierID") > 2;</code>

INNER JOIN

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

Muzisyenler

muzisyenNo	adi	soyadi	ilKodu
9	Ayşe	Yılmaz	00
12	Mehmet	Yorulmaz	06
15	Merve	Sakar	00
18	Hale	Çınar	54
20	Kağan	Yalın	06

İller

ilKodu	ilAdı
00	Bilinmiyor
01	Adana
06	Ankara
34	İstanbul



muzisyenNo	adi	soyadi	ilKodu	ilKodu	ilAdı
9	Ayşe	Yılmaz	00	00	Bilinmiyor
12	Mehmet	Yorulmaz	06	06	Ankara
15	Merve	Sakar	00	00	Bilinmiyor
20	Kaan	Yalın	06	06	Ankara

LEFT JOIN

```
SELECT * FROM "Muzisyenler" LEFT OUTER JOIN "İller"
ON "Muzisyenler"."ilKodu" = "İller"."ilKodu"
```

Muzisyenler

muzisyenNo	adi	soyadi	ilKodu
9	Ayşe	Yılmaz	33
12	Mehmet	Yorulmaz	06
15	Merve	Sakar	00
20	Kağan	Yalın	06
22	Cenk	Dur	07

İller

ilKodu	ilAdı
00	Bilinmiyor
01	Adana
06	Ankara



muzisyenNo	adi	soyadi	ilKodu	ilKodu	ilAdı
9	Ayşe	Yılmaz	33	NULL	NULL
12	Mehmet	Yorulmaz	06	06	Ankara
15	Merve	Sakar	00	00	Bilinmiyor
20	Kaan	Yalın	06	06	Ankara
22	Cenk	Dur	07	NULL	NULL

RIGHT JOIN

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

Muzisyenler

muzisyenNo	adi	soyadi	ilKodu
9	Ayşe	Yılmaz	00
12	Mehmet	Yorulmaz	06
15	Merve	Sakar	00
20	Kağan	Yalın	06

İller

ilKodu	ilAdı
00	Bilinmiyor
01	Adana
06	Ankara



muzisyenNo	adi	soyadi	ilKodu	ilKodu	ilAdı
9	Ayşe	Yılmaz	00	00	Bilinmiyor
15	Merve	Sakar	00	00	Bilinmiyor
NULL	NULL	NULL	NULL	01	Adana
12	Mehmet	Yorulmaz	06	06	Ankara
20	Kaan	Yalın	06	06	Ankara

BÖLÜM 5

```
CREATE DATABASE "AlisVerisUygulamasi"
```

```
CREATE SCHEMA "sema1";
```

```
CREATE TABLE "sema1"."Urunler" (
  "urunNo" SERIAL,
  "kodu" CHAR(6) NOT NULL,
  "adi" VARCHAR(40) NOT NULL,
  "uretimTarihi" DATE DEFAULT '2019-01-01',
  "birimFiyati" MONEY,
  "miktarı" SMALLINT DEFAULT 0,
  CONSTRAINT "urunlerPK" PRIMARY KEY("urunNo"),
```

```

        CONSTRAINT "urunlerUnique" UNIQUE("kodu"),
        CONSTRAINT "urunlerCheck" CHECK("miktar" >= 0)
    );

DROP TABLE "sema1"."Urunler";

DROP SCHEMA "sema1";

DROP DATABASE "AlisVerisUygulaması";

```

TRUNCATE TABLE -> Bir tablonun içindeki tüm verileri silmek için kullanılır.

ALTER TABLE -> Tablonun yapısını düzenlemek için kullanılır. Bir nevi sonradan üzerine ekleme gibi (Migrationa benzetilebilir)

```

ALTER TABLE "Urunler" ADD COLUMN "uretimYeri" VARCHAR(30);
ALTER TABLE "Urunler" DROP COLUMN "uretimYeri";
ALTER TABLE "Urunler" ADD "uretimYeri" VARCHAR(30);
ALTER TABLE "Urunler" ALTER COLUMN "uretimYeri" TYPE CHAR(20);

```

SERIAL -> otomatik artırım sağlar (MSSQL increment benzetilebilir.)

SEQUENCE -> seriala benzetilebilir sonradan bir değişken olarak tanımlanabilir.

```

CREATE SEQUENCE "sayac";
ALTER SEQUENCE "sayac" OWNED BY "Urunler"."urunNo";

```

- SEQUENCE nesnesinin bir sonraki değerini NEXTVAL kullanarak elde edebiliriz.
- NOT NULL olarak tanımlanan kısım boş geçilmez (MSSQL allowed benzetilebilir.)
- DEFAULT kullanılmışsa bir değer atanmadan geçilmiş ise rastgele değer atar.

```

CREATE TABLE "Urunler" (
    "urunNo" SERIAL,
    "kodu" CHAR(6) NOT NULL,
    "adi" VARCHAR(40) NOT NULL,
    "uretimTarihi" DATE DEFAULT '2019-01-01',
    "birimFiyati" MONEY,
    "miktar" SMALLINT DEFAULT 0,
    CONSTRAINT "urunlerPK" PRIMARY KEY("urunNo"),
    CONSTRAINT "urunlerUnique" UNIQUE("kodu"),
    CONSTRAINT "urunlerCheck" CHECK("miktar" >= 0)
);

```

- UNIQUE tanımlandığı alanda tekliği sağlar. Bir tabloda birden fazla olabilir.

```

CREATE TABLE "Urunler" (
    "urunNo" SERIAL,
    "kodu" CHAR(6) NOT NULL,
    "adi" VARCHAR(40) NOT NULL,
    "uretimTarihi" DATE DEFAULT '2019-01-01',
    "birimFiyati" MONEY,
    "miktar" SMALLINT DEFAULT '0',
    CONSTRAINT "urunlerPK" PRIMARY KEY("urunNo"),
    CONSTRAINT "urunlerUnique" UNIQUE("kodu"),
    CONSTRAINT "urunlerCheck" CHECK("miktar" >= 0)
);

```

```
);
```

- CHECK Tanımlandığı alandaki değer aralığını sınırlamada kullanılır.

```
CREATE TABLE "Urunler" (  
    "urunNo" SERIAL,  
    "kodu" CHAR(6) NOT NULL,  
    "adi" VARCHAR(40) NOT NULL,  
    "uretimTarihi" DATE DEFAULT '2019-01-01',  
    "birimFiyati" MONEY,  
    "miktarı" SMALLINT DEFAULT '0',  
    CONSTRAINT "urunlerPK" PRIMARY KEY("urunNo"),  
    CONSTRAINT "urunlerUnique" UNIQUE("kodu"),  
    CONSTRAINT "urunlerCheck" CHECK("miktarı" >= 0)  
);
```

- FOREIGN KEY tanımlanırken bağımlı olduğu şeyleri silerken falan ne yapılacağını belirleyen 3 adet action çeşidimiz vardır. Bunlar silinirken güncellenirken falan kullanılır.

NO ACTION , RESTRICT (kısıtlamak) , CASCADE

```
ALTER TABLE "Urunler"  
ADD CONSTRAINT "urunlerFK" FOREIGN KEY("urunTipi")  
REFERENCES "UrunTipleri"("tipNo")  
ON DELETE NO ACTION  
ON UPDATE NO ACTION;
```

- INDEX belirlenen kısım için sıralanmış indexlenmiş arkaplanda bir yeni tablo oluşturur

```
CREATE INDEX "musterilerAdiIndex" ON "Musteriler" ("adi");  
CREATE INDEX "musterilerSoyadiIndex" ON "Musteriler" USING btree ("soyadi");  
DROP INDEX "musterilerAdiIndex";
```

BÖLÜM 6

- WHERE ifadesinde yalnızca =, !=, <, > gibi operatörler kullanılıyor ise alt sorgular sonucunda tek alan ve tek satır dönmeli ve veri tipi uygun olmalı. Aksi halde hata verir.
- Alt sorgudaki koşul içerisinde birincil anahtar kullanılarak alt sorgudan tek değer döndürülmesi garanti edilebilir.

WHERE İLE TEK DEĞER DÖNDÜREN SORGU

```
SELECT AVG("UnitPrice") FROM "products";  
  
SELECT "ProductID", "UnitPrice" FROM "products"  
WHERE "UnitPrice" < (SELECT AVG("UnitPrice") FROM "products");
```

WHERE İLE ÇOKLU DEĞER DÖNDÜREN SORGU

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

ANY KULLANIMI (HERHANGİ BİRİSİ İÇİN)

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

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

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

```
SELECT * FROM "products"
WHERE "UnitPrice" > ANY
(
    SELECT "UnitPrice"
    FROM "suppliers"
    INNER JOIN "products"
    ON "suppliers"."SupplierID" = "products"."SupplierID"
    WHERE "suppliers"."CompanyName" = 'Tokyo Traders'
);
```

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


ALL KULLANIMI (TÜM DEĞERLER İÇİN)

- > 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'
);
```

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

HAVING İLE ALT SORGU KULLANIMI

- `SELECT AVG("UnitsInStock") FROM "products";`

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

- `SELECT MAX("Quantity") FROM "order_details";`

```
SELECT "ProductID", SUM("Quantity")
FROM "order_details"
GROUP BY "ProductID"
HAVING SUM("Quantity") > (SELECT MAX("Quantity") FROM "order_details");
```

KÜME İŞLEMLERİ

UNION VE UNION ALL (BİRLEŞİM)

- UNION ifadesi ile aynı kayıtlar bir defa gösterilir.
- UNION ALL ifadesi ile aynı kayıtlar gösterilir

```
SELECT "CompanyName", "Country" FROM "customers"  
UNION  
SELECT "CompanyName", "Country" FROM "suppliers"  
ORDER BY 2;
```

```
SELECT "CompanyName", "Country" FROM "customers"  
UNION ALL  
SELECT "CompanyName", "Country" FROM "suppliers"  
ORDER BY 2;
```

INTERSECT (KESİŞİM)

Rasgele iki tablonun kesişimi alınamaz.

- İki tablonun nitelik sayıları aynı olmalı.
- Aynı sıradaki nitelikleri aynı değer alanı üzerinde tanımlanmış olmalı.

```
SELECT "CompanyName", "Country" FROM "customers"  
INTERSECT  
SELECT "CompanyName", "Country" FROM "suppliers"  
ORDER BY 2;
```

EXCEPT (FARK)

Rastgele iki tabloya uygulanamaz.

- İki tablonun nitelik sayıları aynı olmalı.
- Aynı sıradaki nitelikleri aynı değer alanı üzerinde tanımlanmış olmalı.

```
SELECT "CompanyName", "Country" FROM "customers"  
EXCEPT  
SELECT "CompanyName", "Country" FROM "suppliers"  
ORDER BY 2;
```

TRANSACTION

- ACID ifadesi, Atomicity, Consistency, Isolation ve Durability kelimelerinin ilk harflerinin birleşiminden oluşur. Detayları aşağıda anlatılmıştır.
- 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.
- Consistency (Tutarlılık): Herhangi bir kısıt ihlal edilirse roll back işlemiyle veritabanı eski kararlı haline döndürülür.
- Isolation (Yalıtım): İşlemler birbirlerini (ortak kaynak kullanımı durumunda) etkilemezler. Kullanılan ortak kaynak işlem tarafından, işlem tamamlanana kadar, kilitlenir.
- Durability (Mukavemet): Sistem tarafından bir hata meydana gelmesi durumunda tamamlanmamış olan işlem sistem çalışmaya başladıktan sonra mutlaka tamamlanır.

```
BEGIN;

UPDATE Hesap SET bakiye = bakiye - 100.00
  WHERE adi = 'Ahmet';

SAVEPOINT my_savepoint;

UPDATE Hesap SET bakiye = bakiye + 100.00
  WHERE adi = 'Mehmet';

-- Parayı Mehmet'e değil Ayşe'ye gönder

ROLLBACK TO my_savepoint;

UPDATE Hesap SET bakiye = bakiye + 100.00
  WHERE adi = 'Ayşe';

COMMIT;
```

Fonksiyon / Saklı Yordam (Stored Procedure)

Avantajları

- Uygulamanın başarımını iyileştirir.
 - Fonksiyonlar / saklı yordamlar, bir defa oluşturulduktan sonra derlenerek veritabanı kataloğunda saklanır. Her çağrıldıklarında SQL motoru tarafından derlenmek zorunda olan SQL ifadelerine göre çok daha hızlıdır.
- Uygulama ile veritabanı sunucusu arasındaki trafiği azaltır.
 - Uzun SQL ifadeleri yerine fonksiyonun / saklı yordamın adını ve parametrelerini göndermek yeterlidir. Ara sonuçların istemci ve sunucu arasında gönderilmesi önlenir.
- Yeniden kullanılabilir (reusable).
 - Tasarım ve uygulama geliştirme sürecini hızlandırır.
- Güvenliğin sağlanması açısından çok kullanışlıdır.
 - Veritabanı yöneticisi, fonksiyonlara / saklı yordamlara hangi uygulamalar tarafından erişileceğini, tabloların güvenlik düzeyleriyle uğraşmadan, kolayca belirleyebilir.

Dezavantajları

- Fonksiyon / saklı yordam ile program yazmak, değiştirmek (sürüm kontrolü) ve hata bulmak zordur.
- Veritabanı Yönetim Sistemi, veri depolama ve listeleme işlerine ek olarak farklı işler yapmak zorunda da kalacağı için bellek kullanımı ve işlem zamanı açısından olumsuz sonuçlara neden olabilir.
- Fonksiyonların / saklı yordamların yapacağı işler uygulama yazılımlarına da yaptırılabilir.
- Uygulamanın iş mantığı veritabanı sunucusuna kaydırıldığı için uygulama ile veritabanı arasındaki bağımlılık artar ve veritabanından bağımsız kodlama yapmak gitgide imkansızlaşır.

```
CREATE OR REPLACE FUNCTION "fonksiyonTanimlama"(mesaj text,  
altKarakterSayisi SMALLINT, tekrarSayisi integer)  
RETURNS TEXT -- SETOF TEXT, SETOF RECORD diyerek çok sayıda değerin  
döndürülmesi de mümkündür  
AS  
$$  
DECLARE  
    sonuc TEXT; -- Değişken tanımlama bloğu  
BEGIN  
    sonuc := '';  
    IF tekrarSayisi > 0 THEN  
        FOR i IN 1 .. tekrarSayisi LOOP  
            sonuc := sonuc || i || '.' || SUBSTRING(mesaj FROM 1 FOR  
altKarakterSayisi) || E'\r\n';  
            -- E: string içerisindeki (E)scape karakterleri için...  
        END LOOP;  
    END IF;  
    RETURN sonuc;  
END;  
$$  
LANGUAGE 'plpgsql' IMMUTABLE SECURITY DEFINER;
```

Tetikleyici (Trigger)

Avantajları

- Veri bütünlüğünün sağlanması için alternatif bir yoldur. (Örneğin; ürün satıldığında stok miktarının da azaltılması)
- Zamanlanmış görevler için alternatif bir yoldur.
 - Görevler beklenmeden INSERT, UPDATE ve DELETE işlemlerinden önce ya da sonra otomatik olarak yerine getirilebilir.
- Tablolardaki değişikliklerin günlüğünün tutulması (logging) işlemlerinde oldukça faydalıdır.

Dezavantajları

- Veritabanı tasarımının anlaşılabilirliğini düşürür.
 - Fonksiyonlarla / saklı yordamlarla birlikte görünür veritabanı yapısının arkasında başka bir yapı oluştururlar.
- Ek iş yükü oluştururlar ve dolayısıyla işlem gecikmeleri artabilir.
 - Tablolarla ilgili her değişiklikte çalıştıkları için ek iş yükü oluştururlar ve bunun sonucu olarak işlem gecikmeleri artabilir.

```
CREATE TRIGGER "urunBirimFiyatDegistiginde"  
BEFORE UPDATE ON "products"  
FOR EACH ROW  
EXECUTE PROCEDURE "urunDegisikligiTR1"();
```

```
CREATE TRIGGER "kayitKontrol"  
BEFORE INSERT ON "customers" --before ifadesi, veriyi eklemekten önce üzerinde  
işlem yapılabilmesini sağlar  
FOR EACH ROW  
EXECUTE PROCEDURE "kayitEkleTR1"();
```

```
ALTER TABLE "products"  
ENABLE TRIGGER "urunBirimFiyatDegistiginde";
```

```
ALTER TABLE "products"  
DISABLE TRIGGER "urunBirimFiyatDegistiginde";
```

```
DROP TRIGGER "urunBirimFiyatDegistiginde" ON "products";
```

```
DROP TRIGGER IF EXISTS "urunBirimFiyatDegistiginde" ON "products";
```

```
SELECT CURRENT_DATE; -- 2019-11-29 -
```

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

```
SELECT LOCALTIME; -- 10:29:23.910726
```

```
SELECT CURRENT_TIMESTAMP; -- 2019-11-29 10:32:45.738494+03
```

```
SELECT NOW(); -- 2019-11-29 10:32:27.821435+03
```

```
SELECT LOCALTIMESTAMP; -- 2019-11-29 10:33:16.82736
```

```
SELECT AGE(timestamp '2018-04-10', timestamp '1957-06-13'); -- 60 years 9 mons 27 days
```

```
SELECT DATE_PART('years', AGE(timestamp '2000-10-07'));
```

```
SELECT EXTRACT(day from INTERVAL '2 years 5 months 4 days');
```

```
SELECT DATE_TRUNC('minute', timestamp '2018-10-07 23:05:40'); -- 2018-10-07  
23:05:00
```

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

```
SELECT JUSTIFY_HOURS(interval '27 hours'); -- 1 gün 03:00:00
```

```
SELECT JUSTIFY_INTERVAL(interval '1 mon -1 hour') -- 29 gün 23:00:00
```

```
SELECT EXTRACT(EPOCH FROM NOW()); -- 1575267225.07053
```

BÖLÜM 7

NORMALİZASYON

- Normalizasyon, veri fazlalıklarını en aza indirerek veri düzensizliklerinin (data anomaly) önüne geçebilmek için tablo yapılarını değerlendirme ve düzeltme işlemi olarak tanımlanabilir.
- 2NF 1NF den, 3NF 2NF den, ve 4NF 3NF den daha iyidir.
- Her tasarım için en yüksek NF daha iyi sonuç verir denemez. Yüksek başarıma ihtiyaç duyulan bazı durumlarda normal formun (NF) düşürülmesi (denormalizasyon) gerekebilir.
- Normalizasyonun en temel faydası, **veri tekrarı** en aza indirerek veri düzensizliklerinin (data anomaly) önüne geçmemize yardımcı olmasıdır

Veri Tekrarının Zararları

- Veri tekrarı aynı bilgi defalarca kaydedilir.
- Veri tekrarı, kaynak kullanımında israfa yol açar.
- Veri tekrarı, veri tutarsızlıklarına (düzensizliklerine) neden olabilir.
 - Veri girişinde tutarsızlık
 - Veri güncellenmesinde tutarsızlık
 - Veri silmede tutarsızlık

Birinci Normal Form (1NF)

Bir tablonun birinci normal formda (1NF) olması için aşağıdaki şartlar sağlanmalıdır.

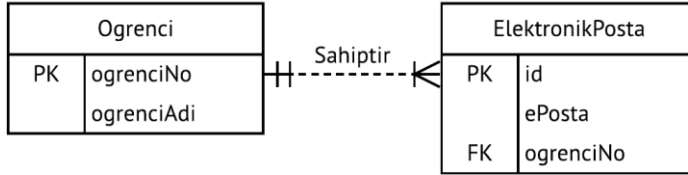
- Tüm alanlar birincil anahtar tarafından belirlenebilmelidir.
- Tüm alanlar tek değerli olmalıdır.
- Aşağıdaki tablo birinci normal formda (1NF) değildir.
 - Çünkü ePosta sütunu tek değerli değildir.
 - Normalizasyon yapmalıyız.

ogrenciNo	ogrenciAdi	ePosta
B05051005	Nagihan Kartal	nkartal@sakarya.edu.tr, nkartal@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

Normalizasyon Sonucu

ogrenciNo	ogrenciAdi
B05051005	Nagihan Kartal
B01031003	Mehmet Arslan
B01032001	Hakan Demir
B03013001	Filiz Şahin

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

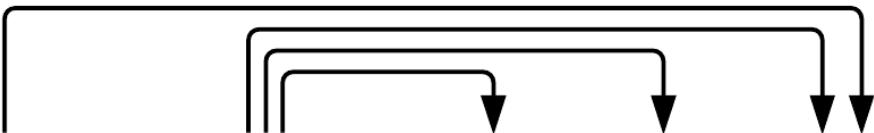


İkinci Normal Form (2NF)

- Kayıtların ayırt edilebilmesi, tabolar arasında ilişki kurulabilmesini sağlar.
- 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**).

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.



<u>ogrenciNo</u>	<u>dersNo</u>	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

Tabloyu 2NF'ye Dönüştürme (Kısmi Bağımlılıkların Giderilmesi)


<u>ogrenciNo</u>	<u>dersNo</u>	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

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

<u>dersNo</u>	dersAdi	kredi
T001	Türkçe	4
T002	Tarih	3
C001	Coğrafya	3

Üçüncü Normal Form (3NF)

- Eğer $A \rightarrow B$ ve $B \rightarrow C$ ise $A \rightarrow B \rightarrow C$
 - A, B üzerinden C'yi belirler.
 - C, A ya geçişken bağımlıdır.
- Aşağıdaki tabloda geçişken bağımlılık vardır.
 - $oduncNo \rightarrow ISBNNo \rightarrow kitapAdi$
 - $oduncNo \rightarrow ISBNNo \rightarrow yayinYili$
 - $oduncNo$ alanı, $ISBNNo$ alanı üzerinden $kitapAdi$ alanını belirler.
 - $kitapAdi$ alanı, $oduncNo$ alanına geçişken bağımlıdır.



<u>oduncNo</u>	uyeNo	ISBNNo	kitapAdi	yayinYili	oduncTarihi	teslimTarihi
1	1000	235540300000	Havuz	2017	2017-07-01	2017-07-10
2	1001	235540300000	Havuz	2017	2017-05-20	2017-06-01
3	1002	958540460098	Gün	2005	2017-11-10	2017-11-15
4	1003	38479960032	Gül	2001	2017-03-20	2017-03-27
5	1001	38479960032	Gül	2001	2017-11-18	2017-11-18

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

<u>oduncNo</u>	uyeNo	ISBNNo	kitapAdi	yayinYili	oduncTarihi	teslimTarihi
1	1000	235540300000	Havuz	2017	2017-07-01	2017-07-10
2	1001	235540300000	Havuz	2017	2017-05-20	2017-06-01
3	1002	958540460098	Gün	2005	2017-11-10	2017-11-15
4	1003	38479960032	Gül	2001	2017-03-20	2017-03-27
5	1001	38479960032	Gül	2001	2017-11-18	2017-11-18

<u>oduncNo</u>	uyeNo	ISBNNo	oduncTarihi	teslimTarihi
1	1000	235540300000	2017-07-01	2017-07-10
2	1001	235540300000	2017-05-20	2017-06-01
3	1002	958540460098	2017-11-10	2017-11-15
4	1003	38479960032	2017-03-20	2017-03-27
5	1001	38479960032	2017-11-18	2017-11-18

<u>ISBNNo</u>	kitapAdi	yayinYili
235540300000	Havuz	2017
958540460098	Gün	2005
38479960032	Gül	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.**

BÖLÜM 8

- EXPLAIN ANALYSE ifadesi ile SQL sorgularının başarımına ilişkin detaylı bilgi edinebiliriz.
- 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.

LIMIT ve OFFSET

- İlk 40 dan sonraki 20 kayıt getirilsin.

```
EXPLAIN ANALYSE
SELECT "store"."store_id", "film"."title"
FROM "inventory"
INNER JOIN "film" ON "inventory"."film_id" = "film"."film_id"
INNER JOIN "store" ON "inventory"."store_id" = "store"."store_id"
LIMIT 20 OFFSET 40;
```

BÖLÜM 9

Veritabanı Güvenliği, Yetkilendirme

- psql, postgresql sunucuya bağlanıp işlemler yapmamızı sağlayan konsol uygulamasıdır.

```
CREATE ROLE "rol1";
CREATE ROLE "rol2" WITH SUPERUSER;
DROP ROLE "rol1";
```

- Roller oluşturulduktan sonra düzenlenebilir.

```
ALTER ROLE "rol1" WITH SUPERUSER CREATEDB;
CREATE USER "kullanici1" WITH PASSWORD 'abc';
```

- CREATE USER ifadesi varsayılan olarak LOGIN yetkili rol oluşturur.
- CREATE ROLE ifadesi varsayılan olarak LOGIN yetkisi olmayan rol oluşturur.
- rol1 isimli rolün yetkilerine gruprol isimli rolün yetkilerini de ekle.

```
GRANT "gruprol" TO "rol1";
```

- Diğer bir deyişle INHERIT yetkisine sahip olması gerekir.
- Bu yetki yoksa, yetkiler kalıtım alınmaz.

- rol1 isimli role verilmiş yetkilerin (gruprol yetkileri) geri alınması.

```
REVOKE "gruprol" FROM "rol1";
```

- rol1 isimli role customers tablosu üzerinde seçim yapma yetkisi ver.

```
GRANT SELECT ON "customers" TO "rol1";
```

- Tüm rollere customers tablosu üzerinde kayıt ekleme yetkisi ver.

```
GRANT INSERT ON "customers" TO PUBLIC;
```

- rol1 isimli kullanıcıya customers tablosu üzerinde tüm yetkileri ver.

```
GRANT ALL ON "customers" TO "rol1";
```

- rol1 isimli rolün customers tablosu üzerindeki güncelleme yetkisini geri al.

```
REVOKE UPDATE ON "customers" FROM "rol1";
```

- rol1 isimli rolün customers tablosu üzerindeki tüm yetkilerini geri al.

```
REVOKE ALL ON "customers" FROM "rol1";
```

- rol1 kullanıcısının Sema1 icerisindeki nesnelere ait tüm yetkileri geri alınır.

```
REVOKE ALL ON SCHEMA "Sema1" FROM "rol1";
```

- Herhangi bir nesne üzerinde yetkiye sahip olan bir rolü silemeyiz.

```
CREATE ROLE "rol1";
```

```
GRANT SELECT ON "customers" TO "rol1";
```