WEB & YAZILIM GELİŞTİRME SERİSİ · MODÜL 8

Veritabanı & SQL

İlişkisel tablolar, veri tipleri ve birincil anahtar; SELECT, WHERE, ORDER BY ile sorgular; INSERT/UPDATE/DELETE, yabancı anahtar ve JOIN'ler; gruplama, normalizasyon, indeksler ve güvenlik. Özgün diyagramlar ve gerçek SQL örnekleriyle.

SELECT · JOIN · Tasarım · Güvenlik · Eğitim amaçlıdır

Bu Kitap Hakkında

Bu modül, verinin kalıcı evi olan ilişkisel veritabanlarını ve onların dili SQL'i sıfırdan öğretir. Dört seviye ve yirmi altı bölüm boyunca tablolardan, veri tiplerinden ve birincil anahtardan SELECT, WHERE ve sıralamaya; veriyi değiştirmekten (INSERT/UPDATE/DELETE) yabancı anahtarlara ve JOIN'lere; gruplama, normalizasyon ve indekslerden güvenlik, işlemler ve tasarıma kadar uzanır.

Her bölümde konuyu görselleştiren özgün bir diyagram (gerçek tablo görselleri, ER ilişki şemaları, JOIN Venn diyagramları), çalıştırılabilir SQL örnekleri, 'sorgu ne döndürür?' kartı ve bir alıştırma yer alır. Güvenlik baştan sona vurgulanır: WHERE'siz UPDATE/DELETE tehlikeleri, SQL enjeksiyonuna karşı parametreli sorgular, işlemlerle (ACID) veri bütünlüğü ve 3-2-1 yedekleme. Bu, on altı modüllük 'Web & Yazılım Geliştirme' serisinin sekizinci modülüdür; buradaki temeller PHP, Python ve C# modüllerinde veritabanına bağlanırken doğrudan kullanılır. Bu seri eğitim amaçlıdır.

Web & Yazılım Geliştirme Serisi · Modül 8

İçindekiler

SEVİYE 1

Veritabanı ve Tablolar

Temeller: veritabanı nedir, ilişkisel tablolar, sütunlar ve veri tipleri, birincil anahtar; ilk sorgular SELECT, WHERE, ORDER BY, LIMIT ve DISTINCT.

BÖLÜM 01

Veritabanı Nedir?

Veritabanı, verinin düzenli, güvenli ve hızlı erişilebilir biçimde saklandığı bir sistemdir. Birkaç satırlık veriyi bir dosyada tutabilirsin; ama binlerce kullanıcı, sipariş ve ürünü güvenle yönetmek için veritabanı gerekir.

Neden dosya değil, veritabanı?

- **Hız:** milyonlarca kayıt arasından saniyede arama.
- **Bütünlük:** kurallar ve ilişkilerle veri tutarlı kalır.
- **Eşzamanlılık:** birçok kullanıcı aynı anda güvenle erişir.
- **Güvenlik:** kimin neye erişebileceği denetlenir.

Şema 1.1 — Veri büyüdükçe dosya yetmez; veritabanı gerekir.

İPUCU

Bu modülde **ilişkisel veritabanlarına** ve onların dili olan **SQL**'e odaklanacağız — bunlar sektörün en yaygın temelidir. SQL (Structured Query Language), neredeyse tüm ilişkisel veritabanlarında (MySQL, PostgreSQL, SQLite, SQL Server) benzer biçimde çalışır; bir kez öğrenince hepsine yakın olursun. Backend modülünde gördüğün "sunucu veritabanına bakar" cümlesinin somut hâli budur.

Sorgu ne döndürür? SONUÇ

Bir veritabanına "kayıtlı kullanıcıların listesini ver" gibi bir **sorgu** gönderirsin; veritabanı bu sorguya bir **sonuç** (genelde bir satır kümesi / tablo) döner. SQL, bu sorguları yazdığın dildir.

Alıştırma

8 dk

Veritabanını kavra:

1. Bir alışveriş sitesinin hangi verileri saklaması gerektiğini listele.
2. Bu veriyi bir dosyada tutmanın üç zorluğunu yaz.
3. SQL'in ne işe yaradığını kendi cümlenle açıkla.

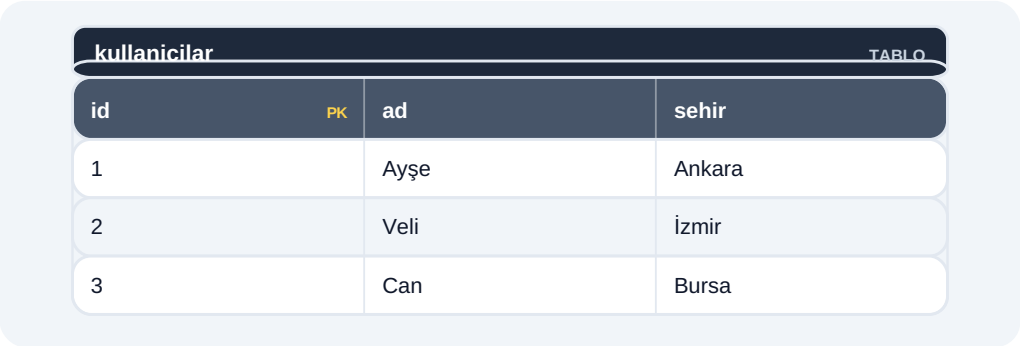
BÖLÜM 02

İlişkisel Veritabanı ve Tablolar

İlişkisel veritabanında veri, tablolarda tutulur. Bir tablo, satırlardan (kayıtlar) ve sütunlardan (alanlar) oluşur — tıpkı bir elektronik tablo gibi, ama kurallar ve ilişkilerle. Her tablo tek bir "şeyi" temsil eder: kullanıcılar, ürünler, siparişler.

Tablonun yapısı

- **Sütun (column):** bir alan (örn. ad, eposta) — dikey.
- **Satır (row):** bir kayıt (bir kullanıcı) — yatay.
- **Hücre:** bir satır ve sütunun kesişimi (bir değer).



Şema 2.1 — Bir tablo: sütunlar (alanlar) ve satırlar (kayıtlar).

İPUCU

Her tabloya tek bir "varlık" düşür: kullanıcılar bir tabloda, siparişler ayrı bir tabloda. Bir tabloya birbiriyle ilgisiz şeyleri (kullanıcılar + ürünler) karıştırmak, ileride büyük baş ağrısı yaratır. Tablo isimleri genelde **çoğul** ve küçük harftir (`kullanicilar` , `urunler`). İyi tablo tasarımı, iyi veritabanının temelidir.

Sorgu ne döndürür? SONUÇ

`kullanicilar` tablosundan veri istediğinde, veritabanı sana satırlardan oluşan bir sonuç döner. Her satır bir kullanıcıyı, her sütun o kullanıcının bir özelliğini (ad, şehir) temsil eder.

Alıştırma

10 dk

Tablo tasarla:

1. Bir kütüphane için "kitaplar" tablosunun sütunlarını belirle.
2. Üç örnek satır (kayıt) yaz.
3. Hangi verilerin ayrı bir tabloda (örn. "yazarlar") olması gerektiğini düşün.

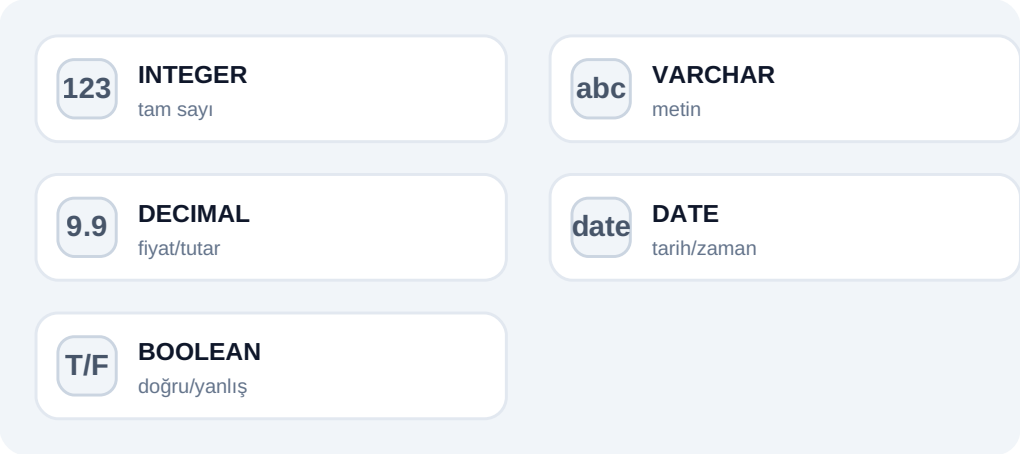
BÖLÜM 03

Sütunlar ve Veri Tipleri

Her sütunun bir veri tipi vardır: o sütunda ne tür değerlerin tutulacağını belirler. Doğru tip seçmek, veriyi hem doğru saklar hem de hatalara karşı korur.

Yaygın veri tipleri

- **INTEGER:** tam sayı (yaş, adet, id).
- **VARCHAR / TEXT:** metin (ad, açıklama).
- **DECIMAL:** ondalıklı sayı (fiyat, tutar).
- **DATE / TIMESTAMP:** tarih ve zaman.
- **BOOLEAN:** doğru/yanlış (aktif mi?).



Şema 3.1 — Sık kullanılan SQL veri tipleri.

Tablo oluşturma (CREATE TABLE)

```
CREATE TABLE urunler (
  id        INTEGER PRIMARY KEY,
  ad        VARCHAR(100),
  fiyat     DECIMAL(10,2),
  stokta    BOOLEAN,
  eklenme   DATE
);
```

İPUCU

Veri tipini doğru seç: parayı **asla** ondalık hatalarına açık türlerle değil, `DECIMAL` ile sakla; tarihleri metin yerine `DATE` olarak tut (böylece sıralama ve karşılaştırma doğru çalışır). Doğru tip, veritabanının senin için bir "bekçi" gibi çalışmasını sağlar: yanlış türde veri girmeye çalışırsan reddeder, böylece veri temiz kalır.

Sorgu ne döndürür? SONUÇ

Bir sütunu `INTEGER` tanımlarsan, oraya metin koymaya çalışınca veritabanı hata verir ve reddeder. Bu sayede "yaş" sütununda asla "yirmi" gibi bir metin bulunmaz — veri tipleri, veriyi baştan doğru tutmanı sağlar.

Alıştırma 10 dk

Tip seç:

1. Bir "öğrenciler" tablosunun her sütunu için uygun veri tipini yaz.
2. Fiyat için neden DECIMAL kullanılır, açıkla.
3. Bir CREATE TABLE ifadesi yaz (en az 4 sütun).

BÖLÜM 04

Birincil Anahtar (Primary Key)

Her tabloda, her satırı benzersiz biçimde tanımlayan bir sütun olmalıdır: birincil anahtar (primary key). Genelde bu, otomatik artan bir "id" sütunudur. Birincil anahtar, bir satıra kesin olarak işaret etmenin yoludur.

Birincil anahtarın kuralları

- **Benzersiz:** iki satırda aynı değer olamaz.
- **Boş olamaz:** her satırın bir anahtarı olmalı.
- **Değişmez:** bir kez atanınca değiştirilmez.

kullanicilar TABLO

id PK	ad	eposta
1	Ayşe	ayse@ornek.com
2	Veli	veli@ornek.com
3	Can	can@ornek.com

Şema 4.1 — "id" sütunu birincil anahtardır (PK): her satırı benzersiz tanımlar.

İPUCU

Neden ad veya e-posta yerine bir `id`? Çünkü adlar tekrar edebilir (iki "Ayşe Yılmaz"), e-postalar değişebilir. Sayısal, otomatik artan bir `id` ise her zaman benzersiz ve değişmezdir — bir satıra güvenle işaret etmenin tek sağlam yolu odur. Birincil anahtar, bir sonraki bölümlerde tabloları birbirine bağlamanın da (yabancı anahtar) temeli olacak.

Sorgu ne döndürür? SONUÇ

`id = 2` dediğinde, veritabanı tam olarak tek bir satırı (Veli) döner — başka hiçbir satırın id'si 2 olamaz. Birincil anahtar, "hangi satır?" sorusuna her zaman tek ve kesin bir cevap verir.

Alıştırma

8 dk

Anahtar seç:

1. Bir "arabalar" tablosu için iyi bir birincil anahtar öner.
2. Neden "marka" sütununun birincil anahtar olamayacağını açıkla.
3. Birincil anahtarın üç kuralını kendi örneğinde göster.

BÖLÜM 05

İlk Sorgu: SELECT

SELECT, SQL'in en temel ve en sık kullanılan komutudur: bir tablodan veri okur. "Şu sütunları, şu tablodan getir" dersin; veritabanı sana sonuç olarak satırları döner.

SELECT'in yapısı

- `SELECT sütunlar` — hangi alanları istiyorsun.
- `FROM tablo` — hangi tablodan.
- `SELECT *` — tüm sütunları getir.

Temel SELECT sorguları

```
 -- Tüm sütunları, tüm satırları getir:
SELECT * FROM kullanicilar;

-- Yalnızca ad ve şehir sütununu getir:
SELECT ad, sehir FROM kullanicilar;
```

sonuç: SELECT ad, sehir FROM kullanicilar TABLO

ad	sehir
Ayşe	Ankara
Veli	İzmir
Can	Bursa

Şema 5.1 — SELECT ad, sehir → yalnızca o iki sütun döner.

İPUCU

Üretimde `SELECT *` yerine **ihtiyacın olan sütunları açıkça yaz**: bu hem daha hızlıdır (gereksiz veri taşınmaz) hem de tabloya sütun eklendiğinde sorgun beklenmedik biçimde değişmez. SQL komutlarının sonuna noktalı virgül (`;`) koymak gelenektir. SQL büyük/küçük harfe duyarsızdır ama anahtar kelimeleri (SELECT, FROM) BÜYÜK yazmak okunabilirliği artırır.

Sorgu ne döndürür? SONUÇ

`SELECT ad, sehir FROM kullanicilar` sorgusu, tablodaki her satır için yalnızca **ad** ve **sehir** sütunlarını içeren bir sonuç döner — id ve e-posta gibi istemediğin alanlar gelmez. İstediğin sütunları sen belirlersin.

Alıştırma 10 dk

SELECT yaz:

1. Bir "urunler" tablosundan tüm sütunları getiren sorguyu yaz.
2. Yalnızca ad ve fiyat sütununu getiren sorguyu yaz.
3. SELECT * ile sütunları tek tek yazmanın farkını açıkla.

BÖLÜM 06

Süzme: WHERE

Çoğu zaman tüm satırları değil, bir koşulu sağlayanları istersin: "şehri Ankara olanlar", "fiyatı 100'den büyük ürünler". WHERE komutu, sonucu bir koşula göre süzer.

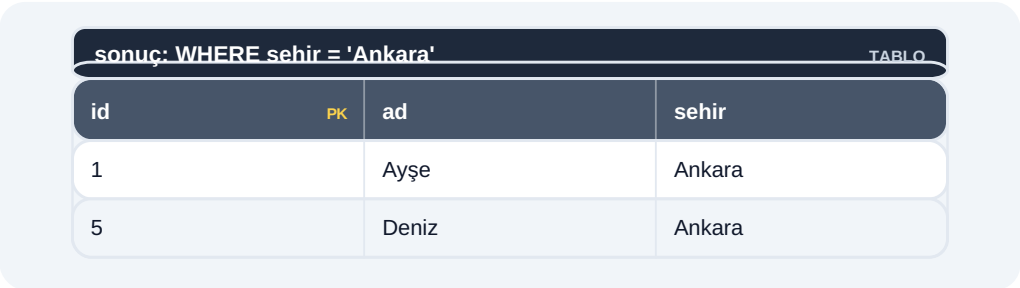
WHERE ile koşul

- Karşılaştırma: `=`, `>`, `<`, `>=`, `<=`, `<>` (eşit değil).
- Birleştirme: `AND`, `OR`, `NOT`.
- Metin arama: `LIKE '%ankara%'`; aralık: `BETWEEN`; liste: `IN`.

WHERE ile süzme

```
 -- Şehri Ankara olan kullanıcılar:
SELECT ad FROM kullanicilar WHERE sehir = 'Ankara';

-- Fiyatı 100'den büyük ürünler:
SELECT ad, fiyat FROM urunler WHERE fiyat > 100;
```



Şema 6.1 — WHERE, yalnızca koşulu sağlayan satırları döndürür.

İPUCU

SQL'de metin değerleri **tek tırnak** içinde yazılır (`'Ankara'`), sayılar tırnaksız. En sık hata, eşitlik için `==` kullanmaktır — SQL'de eşitlik tek `=` ile yazılır. Bir WHERE koşulunu yanlış kurmak (veya unutmak) yanlış satırların gelmesine yol açar; özellikle UPDATE ve DELETE'te bu çok tehlikelidir (sonraki seviyede göreceğiz).

Sorgu ne döndürür? SONUÇ

`WHERE sehir = 'Ankara'` koşulu, yalnızca şehri tam olarak "Ankara" olan satırları döner; diğerleri sonuçta yer almaz. Koşulu değiştirerek (örn. `fiyat > 100`) farklı süzme sonuçları elde edersin.

Alıştırma

12 dk

WHERE yaz:

1. "Yaşı 18'den büyük kullanıcılar" sorgusunu yaz.
2. "Stokta olan ve fiyatı 50'den az ürünler" için AND kullan.
3. Metin değerlerinin neden tırnak içinde yazıldığını açıkla.

BÖLÜM 07

Sıralama ve Sınırlama: ORDER BY, LIMIT

Sonuçları belirli bir düzende görmek (en pahalıdan ucuza, A'dan Z'ye) ve yalnızca ilk birkaçını almak çok yaygındır. ORDER BY sıralar, LIMIT ise sonuç sayısını kısıtlar.

Sıralama ve sınırlama

- `ORDER BY sütun` — artan (ASC, varsayılan).
- `ORDER BY sütun DESC` — azalan.
- `LIMIT n` — yalnızca ilk n satır.

ORDER BY ve LIMIT

```
 -- En pahalı 3 ürün:
SELECT ad, fiyat FROM urunler
ORDER BY fiyat DESC
LIMIT 3;
```

sonuç: ORDER BY fiyat DESC LIMIT 3 — TABLO

ad	fiyat
Dizüstü	24000
Telefon	18000
Tablet	9000

Şema 7.1 — En pahalıdan ucuza sıralanmış, ilk 3 satır.

İPUCU

`ORDER BY` ve `LIMIT` birlikte çok güçlüdür: "en yeni 10 yorum", "en çok satan 5 ürün" gibi sorgular bu ikilisiyle yazılır. Sıralama olmadan `LIMIT` kullanmak **rastgele** satırlar getirebilir — "ilk 3" demek için önce neye göre sıralandığını belirtmelisin. Sıra: önce `ORDER BY`, sonra `LIMIT`.

Sorgu ne döndürür? SONUÇ

`ORDER BY fiyat DESC LIMIT 3`, önce ürünleri fiyata göre en pahalıdan ucuza sıralar, sonra yalnızca ilk 3'ünü döner. ORDER BY olmadan LIMIT, hangi 3 satırın geleceğini garanti etmez.

Alıştırma 10 dk

Sırala ve sınırla:

1. Kullanıcıları ada göre A'dan Z'ye sıralayan sorguyu yaz.
2. "En yeni 5 kayıt" için ORDER BY ve LIMIT kullan.
3. ORDER BY olmadan LIMIT kullanmanın riskini açıkla.

BÖLÜM 08

Tekil Değerler: DISTINCT

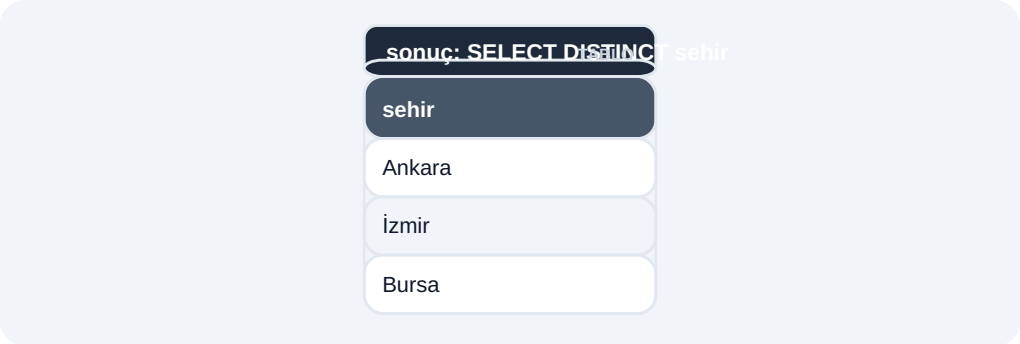
Bazen bir sütundaki tekrar eden değerleri değil, yalnızca benzersiz olanları görmek istersin: "kaç farklı şehir var?" gibi. DISTINCT, sonuçtaki yinelenen satırları teke indirir.

DISTINCT ile tekilleştirme

- `SELECT DISTINCT sütun` — benzersiz değerleri getirir.
- Tekrar eden değerler tek satıra iner.
- "Kaç farklı X var?" sorularının ilk adımı.

DISTINCT ile benzersiz değerler

```
-- Kullanıcıların bulunduğu farklı şehirler:
SELECT DISTINCT sehir FROM kullanicilar;
```



Şema 8.1 — DISTINCT: tekrar eden şehirler teke iner, yalnızca benzersizler kalır.

İPUCU

Tabloda "Ankara" beş kez geçse bile, `SELECT DISTINCT sehir` onu sonuçta yalnızca bir kez gösterir. DISTINCT, "elimde kaç farklı kategori/şehir/durum var?" gibi sorulara hızlı cevap verir. Bir sonraki seviyede bunu sayma (`COUNT`) ve gruplama (`GROUP BY`) ile birleştirerek çok daha güçlü özetler çıkaracaksın.

Sorgu ne döndürür? SONUÇ

Tabloda üç kullanıcı Ankara'da olsa bile, `SELECT DISTINCT sehir` "Ankara"yı yalnızca bir kez döner. Sonuç, sütundaki **benzersiz** değerlerin listesidir — kaç farklı şehir olduğunu bir bakışta görürsün.

Alıştırma

8 dk

DISTINCT kullan:

1. Bir "siparisler" tablosundan farklı ödeme yöntemlerini getiren sorguyu yaz.
2. DISTINCT'in tekrarları nasıl ele aldığını açıkla.
3. "Kaç farklı ülke var?" sorusuna giden ilk adımı yaz.

SEVİYE 2

Veriyi Değiştirmek ve İlişkiler

Veriyi yönetmek ve bağlamak: INSERT, UPDATE, DELETE; yabancı anahtar ile ilişkiler; tabloları JOIN ile birleştirmek ve JOIN türleri.

BÖLÜM 09

Veri Eklemek: INSERT

Bir tabloya yeni satır eklemek için INSERT komutu kullanılır. Hangi sütunlara hangi değerleri koyacağını belirtirsin; veritabanı yeni kaydı tabloya ekler.

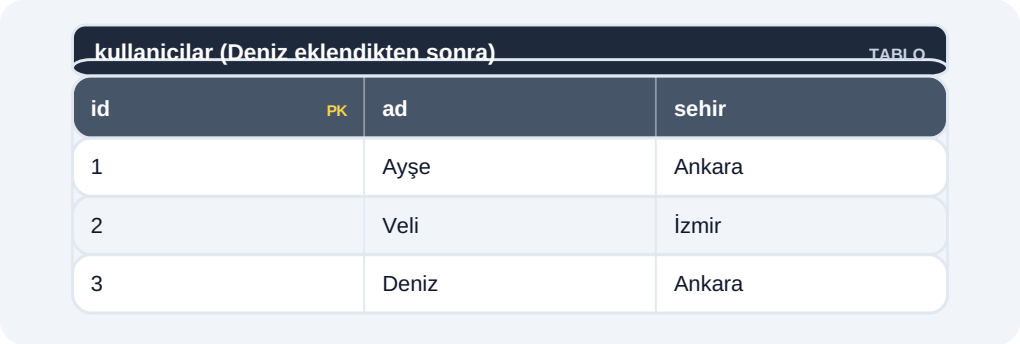
INSERT'in yapısı

- `INSERT INTO tablo (sütunlar)` — hangi alanlar.
- `VALUES (değerler)` — sırasıyla değerler.
- Birincil anahtar genelde otomatik atanır.

INSERT ile satır ekleme

```
 INSERT INTO kullanicilar (ad, sehir, eposta)
VALUES ('Deniz', 'Ankara', 'deniz@ornek.com');

-- Birden çok satır:
INSERT INTO kullanicilar (ad, sehir) VALUES
  ('Ece', 'İzmir'),
  ('Mert', 'Bursa');
```



Şema 9.1 — INSERT sonrası tabloya yeni bir satır eklenir.

İPUCU

Sütun adlarını **açıkça yazmak** iyi bir alışkanlıktır (`INSERT INTO ... (ad, sehir)`): böylece tabloya yeni sütun eklendiğinde sorgun bozulmaz ve hangi değerin nereye gittiği nettir. Değerler, belirttiğin sütun sırasına birebir uymalıdır. Metinler tek tırnakta, sayılar tırnaksız yazılır — Bölüm 6'daki kuralla aynı.

Sorgu ne döndürür? SONUÇ

`INSERT` komutu bir sonuç kümesi döndürmez; bunun yerine tabloyu değiştirir (yeni satır ekler). İşlem sonrası `SELECT * FROM kullanicilar` çalıştırırsan, yeni eklediğin satırı diğerleriyle birlikte görürsün.

Alıştırma 10 dk

INSERT yaz:

1. Bir "urunler" tablosuna yeni bir ürün ekleyen INSERT yaz.
2. Tek bir komutla iki ürün birden ekle.
3. Sütun adlarını açıkça yazmanın faydasını açıkla.

BÖLÜM 10

Veri Güncellemek: UPDATE

Var olan satırları değiştirmek için UPDATE kullanılır: bir fiyatı güncellemek, bir adresi düzeltmek gibi. En kritik nokta, hangi satırların güncelleneceğini WHERE ile doğru belirlemektir.

UPDATE'in yapısı

- `UPDATE tablo SET sütun = değer` — neyi değiştir.
- `WHERE koşul` — **hangi satırlarda** (çok önemli!).
- WHERE unutulursa: **tüm satırlar** değişir.

UPDATE ile güncelleme

```
 -- Doğru: yalnızca id = 3 olan satırı güncelle
UPDATE kullanicilar
SET sehir = 'İstanbul'
WHERE id = 3;

-- TEHLİKELİ: WHERE yok -> herkesin şehri değişir!
UPDATE kullanicilar SET sehir = 'İstanbul';
```

Şema 10.1 — UPDATE'te WHERE, hangi satırların değişeceğini belirler.

İPUCU

UPDATE'i her zaman WHERE ile sınırla. WHERE olmadan çalıştırılan bir UPDATE, tablodaki **tüm satırları** değiştirir — bu, gerçek projelerde yaşanan en yıkıcı hatalardan biridir. Güvenli alışkanlık: önce aynı WHERE ile bir `SELECT` çalıştırıp hangi satırların etkileneceğini gör, sonra UPDATE'i yaz. Bir sonraki seviyede göreceğin "işlemler" (transactions) bu tür hataları geri almanı sağlar.

Sorgu ne döndürür? SONUÇ

`UPDATE ... WHERE id = 3`, yalnızca 3 numaralı satırı günceller ve kaç satırın etkilendiğini bildirir (örn. "1 satır güncellendi"). WHERE'i kaldırırsan bu sayı tüm tablo olur — bu yüzden çalıştırmadan önce iki kez düşün.

Alıştırma 12 dk

UPDATE yaz:

1. Bir ürünün fiyatını güncelleyen, yalnızca o ürünü etkileyen UPDATE yaz.
2. WHERE'siz bir UPDATE'in neden tehlikeli olduğunu açıkla.
3. Güvenli güncelleme için önce hangi sorguyu çalıştırırsın?

BÖLÜM 11

Veri Silmek: DELETE

Satırları silmek için DELETE kullanılır. UPDATE gibi, DELETE de WHERE ile sınırlanmalıdır — yoksa tablodaki tüm satırlar silinir. Silme işlemi çoğu zaman geri alınamaz, bu yüzden en dikkatli yaklaşılması gereken komuttur.

DELETE'in yapısı

- `DELETE FROM tablo WHERE koşul` — eşleşenleri sil.
- WHERE unutulursa: **tüm satırlar silinir**.
- Silme genelde kalıcıdır; geri alınamaz.

DELETE ile silme

```
 -- Doğru: yalnızca id = 5 olan satırı sil
DELETE FROM kullanicilar WHERE id = 5;

-- TEHLİKELİ: WHERE yok -> TÜM kullanıcılar silinir!
DELETE FROM kullanicilar;
```

Şema 11.1 — DELETE'te WHERE hayati önemdedir: yanlışı silmek kalıcıdır.

İPUCU

DELETE, SQL'in en tehlikeli komutudur: yanlış (veya eksik WHERE'li) bir DELETE, saniyeler içinde tüm veriyi yok edebilir. Korunma yolları: **(1)** önce aynı WHERE ile SELECT çalıştır, **(2)** kritik işlemleri bir "işlem" (transaction) içinde yap (gerekirse ROLLBACK), **(3)** düzenli yedek al (Bölüm 24). Birçok sistem, gerçekten silmek yerine "silindi" işareti koyar (soft delete) — böylece veri kurtarılabilir.

Sorgu ne döndürür? SONUÇ

`DELETE FROM kullanicilar WHERE id = 5` yalnızca tek satırı siler. WHERE olmadan aynı komut tüm tabloyu boşaltır — ve çoğu durumda bunu geri almanın tek yolu bir yedektir. Bu yüzden DELETE'ten önce her zaman dur ve WHERE'ini kontrol et.

Alıştırma 10 dk

DELETE yaz:

1. Tek bir siparişi silen, güvenli bir DELETE yaz.
2. WHERE'siz DELETE'in sonucunu açıkla.
3. Gerçek silme yerine "soft delete" yaklaşımını anlat.

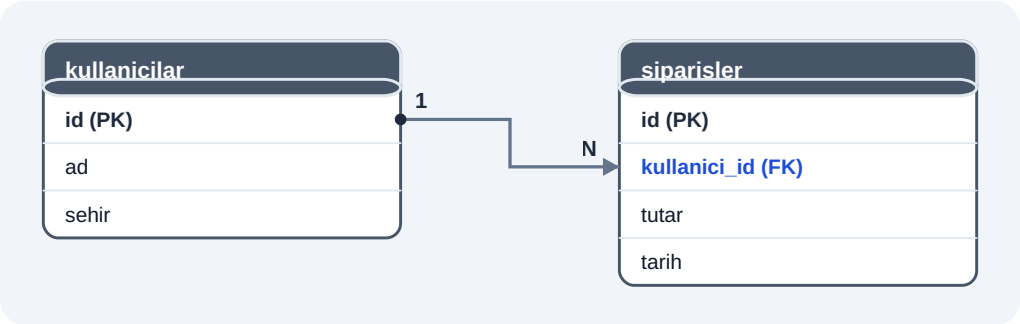
BÖLÜM 12

İlişkiler ve Yabancı Anahtar (Foreign Key)

İlişkisel veritabanının gücü, tabloları birbirine bağlamasıdır. Bir tablodaki yabancı anahtar (foreign key), başka bir tablodaki birincil anahtara işaret eder — böylece "bu sipariş hangi kullanıcıya ait?" sorusu yanıtlanabilir.

Tabloları bağlamak

- **Birincil anahtar (PK):** bir satırı tanımlar (örn. kullanicilar.id).
- **Yabancı anahtar (FK):** başka tablonun PK'sine işaret eder.
- Böylece veri tekrarı önlenir, ilişkiler kurulur.



Şema 12.1 — siparisler.kullanici_id, kullanicilar.id'ye işaret eder (1 kullanıcı → N sipariş).

İPUCU

Yabancı anahtar, veri tekrarını önler ve **bütünlüğü korur**: her siparişte kullanıcının tüm bilgilerini tekrar yazmak yerine, sadece `kullanici_id`'sini tutarsın. Çoğu veritabanı, var olmayan bir kullanıcıya işaret eden bir sipariş eklemeni de engeller (referans bütünlüğü). Bu ilişki türü ("bir kullanıcının çok siparişi") en yaygın olanıdır: **bir-çok** (1→N).

Sorgu ne döndürür? SONUÇ

`siparisler` tablosundaki bir satırın `kullanici_id = 1` değeri, o siparişin `kullanicilar` tablosundaki 1 numaralı satıra (Ayşe) ait olduğunu söyler. Bu bağ sayesinde, bir sonraki bölümde iki tabloyu birleştirip "Ayşe'nin siparişlerini" tek sorguda getirebileceksin.

Alıştırma

12 dk

İlişki tasarla:

1. "yazarlar" ve "kitaplar" tabloları arasında bir ilişki kur.
2. Hangi tabloya yabancı anahtar koyarsın? Neden?
3. Yabancı anahtarın veri tekrarını nasıl önlediğini açıkla.

BÖLÜM 13

Tabloları Birleştirmek: JOIN

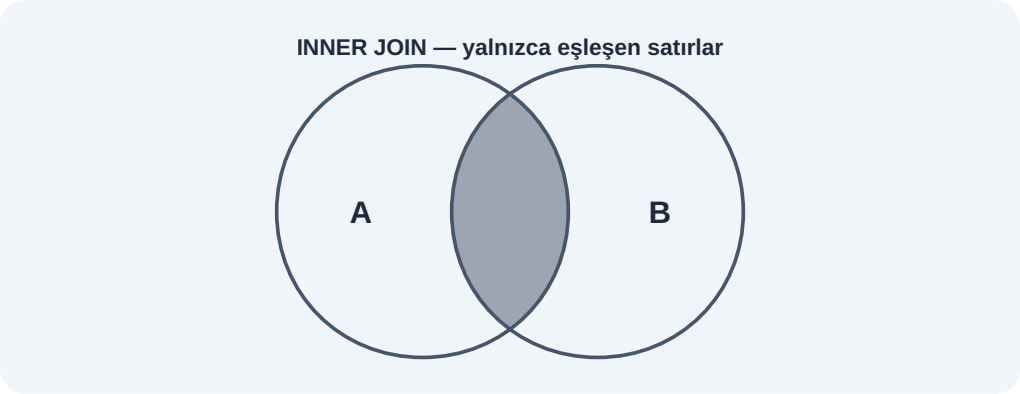
İlişkili veriyi tek bir sonuçta görmek için tabloları JOIN ile birleştirirsin: "her siparişi, ait olduğu kullanıcının adıyla birlikte göster". JOIN, ilişkisel veritabanını gerçekten güçlü kılan komuttur.

INNER JOIN

- İki tabloyu, eşleşen bir sütun (PK = FK) üzerinden birleştirir.
- `ON` ile eşleşme koşulu belirtilir.
- INNER JOIN: yalnızca **her iki tabloda da eşleşen** satırlar.

INNER JOIN ile birleştirme

```
SELECT siparisler.id, kullanicilar.ad, siparisler.tutar
FROM siparisler
INNER JOIN kullanicilar
  ON siparisler.kullanici_id = kullanicilar.id;
```



Şema 13.1 — INNER JOIN: yalnızca iki tabloda da karşılığı olan satırlar döner.

İPUCU

JOIN'in kalbi **ON koşuludur**: hangi sütunların eşleşeceğini söyler (genelde bir tablonun FK'si = diğerinin PK'si). Sütun adları iki tabloda da aynıysa (örn. ikisinde de `id`), hangi tablonunkini kastettiğini `tablo.sutun` ile belirtmelisin. INNER JOIN, eşleşmeyen satırları (örn. hiç siparişi olmayan kullanıcı) sonuçtan dışlar — onları da istiyorsan LEFT JOIN gerekir (sonraki bölüm).

Sorgu ne döndürür? SONUÇ

Bu INNER JOIN, her siparişi ait olduğu kullanıcının adıyla eşleştirip tek bir tabloda döner: sipariş id, kullanıcı adı, tutar. Yalnızca geçerli bir kullanıcıya bağlı siparişler görünür — kullanıcısı olmayan (ya da hiç siparişi olmayan kullanıcı) bu sonuçta yer almaz.

Alıştırma 14 dk

JOIN yaz:

1. "kitaplar" ve "yazarlar" tablolarını yazar adıyla birlikte getiren JOIN yaz.
2. ON koşulunun hangi sütunları eşleştirdiğini belirt.
3. INNER JOIN'in hangi satırları dışladığını açıkla.

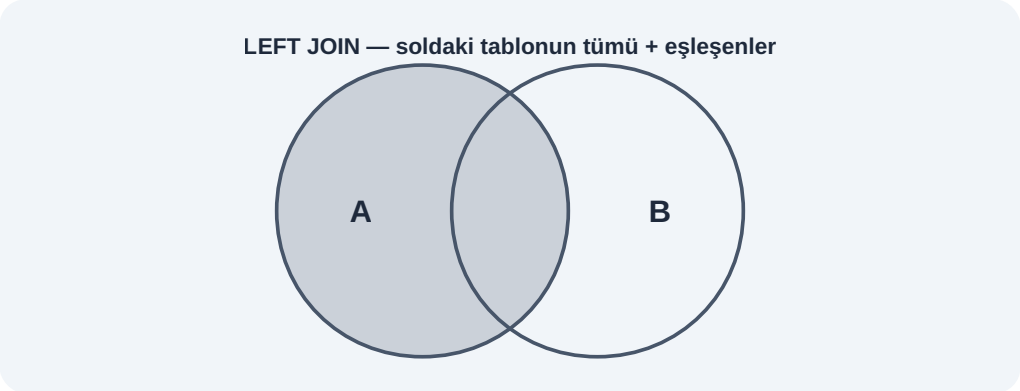
BÖLÜM 14

JOIN Türleri

INNER JOIN yalnızca eşleşenleri getirir; ama bazen "siparişi olmasa bile tüm kullanıcıları" görmek istersin. Farklı JOIN türleri, hangi eşleşmeyen satırların da dahil edileceğini belirler.

Dört temel JOIN türü

- **INNER JOIN:** yalnızca her iki tabloda eşleşenler.
- **LEFT JOIN:** soldaki tablonun tümü + eşleşenler.
- **RIGHT JOIN:** sağdaki tablonun tümü + eşleşenler.
- **FULL JOIN:** her iki tablonun tüm satırları.



Şema 14.1 — LEFT JOIN: soldaki tablonun her satırı gelir; sağda eşleşme yoksa NULL döner.

LEFT JOIN örneği

```
 -- Tüm kullanıcılar; siparişi olmayanların tutarı NULL olur
SELECT kullanicilar.ad, siparisler.tutar
FROM kullanicilar
LEFT JOIN siparisler
  ON kullanicilar.id = siparisler.kullanici_id;
```

İPUCU

Türü ihtiyacına göre seç: "yalnızca sipariş vermiş kullanıcılar" → INNER; "tüm kullanıcılar, sipariş vermemiş olsalar bile" → LEFT JOIN. LEFT JOIN'de eşleşme bulunmayan yerlerde değerler **NULL** gelir (örn. siparişi olmayan kullanıcının tutarı NULL). Pratikte en sık INNER ve LEFT JOIN kullanılır; RIGHT ve FULL daha nadirdir (ve RIGHT, tabloların yerini değiştirerek LEFT ile yazılabilir).

Sorgu ne döndürür? SONUÇ

Bu LEFT JOIN, hiç siparişi olmayan kullanıcıları da sonuca dahil eder — onların `tutar` alanı NULL görünür. Aynı sorguyu INNER JOIN ile yazsaydın, siparişi olmayan kullanıcılar tamamen kaybolurdu. JOIN türü, "eksik eşleşmelerle ne yapılacağını" belirler.

Alıştırma 12 dk

Tür seç:

1. "Hiç kitap yazmamış yazarları da listele" için hangi JOIN türü?
2. LEFT JOIN'de eşleşmeyen satırlarda hangi değer görünür?
3. RIGHT JOIN'in neden LEFT ile yazılabileceğini açıkla.

SEVİYE 3

Özetleme ve Tasarım

Veriden anlam çıkarmak: GROUP BY ile gruplama, toplama fonksiyonları, HAVING; normalizasyon, indeksler ve veritabanı tasarımı.

BÖLÜM 15

Gruplama: GROUP BY

Tek tek satırlar yerine özetler istediğinde GROUP BY kullanırsın: "her şehirde kaç kullanıcı var?", "her kategorinin toplam satışı ne?". GROUP BY, satırları ortak bir değere göre gruplar ve her grup için tek bir özet satır üretir.

GROUP BY mantığı

- Satırları bir sütunun değerine göre gruplar (örn. her şehir).
- Her grup için bir toplama fonksiyonu hesaplanır (COUNT, SUM...).
- Sonuç: her grup için tek bir özet satır.

GROUP BY ile gruplama

```
-- Her şehirde kaç kullanıcı var?
SELECT sehir, COUNT(*) AS adet
FROM kullanicilar
GROUP BY sehir;
```

sonuç: GROUP BY sehir — TABLO

sehir	adet
Ankara	3
İzmir	2
Bursa	1

Şema 15.1 — GROUP BY sehir: her şehir tek satıra iner, yanında sayısıyla.

İPUCU

GROUP BY'ın temel kuralı: SELECT'te yer alan her sütun ya **GROUP BY'da olmalı** ya da bir toplama fonksiyonu içinde kullanılmalıdır. Örneğin `sehir`'e göre grupluyorsan, SELECT'te `sehir` (gruplanan) ve `COUNT(*)` (toplama) olabilir ama tek bir kullanıcının `ad`'ı olamaz — çünkü bir grupta birçok ad vardır. `AS` ile sonuç sütununa anlamlı bir isim verebilirsin.

Sorgu ne döndürür? SONUÇ

`GROUP BY sehir`, aynı şehirdeki tüm satırları tek bir grupta toplar; `COUNT(*)` her grubun kaç satır içerdiğini sayar. Sonuçta her şehir bir kez görünür, yanında o şehirdeki kullanıcı sayısıyla — ham satırlar yerine bir özet elde edersin.

Alıştırma

12 dk

GROUP BY yaz:

1. "Her kategoride kaç ürün var?" sorgusunu yaz.
2. Sonucun kaç satır olacağını tahmin et (kategori sayısı kadar).
3. GROUP BY kuralını (SELECT'teki sütunlar) açıkla.

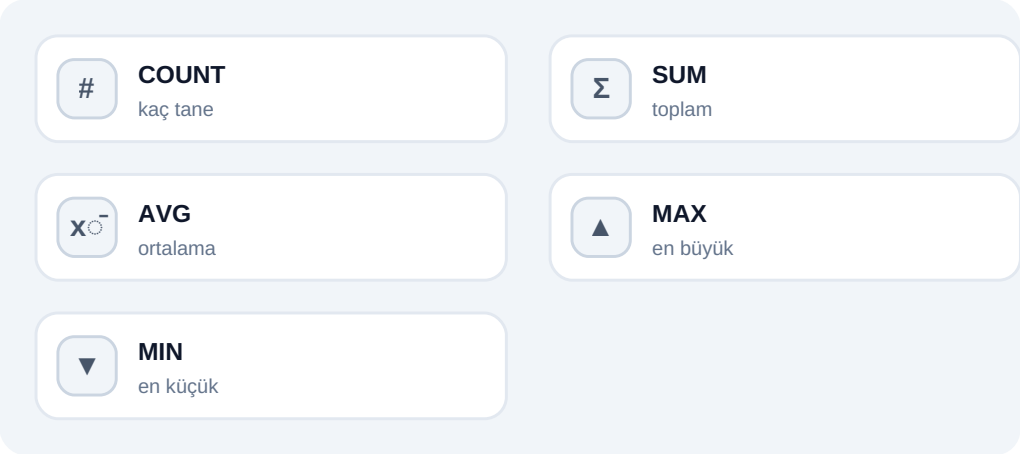
BÖLÜM 16

Toplama Fonksiyonları

Toplama (aggregate) fonksiyonları, bir grup satırı tek bir değere indirir: kaç tane (COUNT), toplam (SUM), ortalama (AVG), en büyük (MAX), en küçük (MIN). GROUP BY ile birlikte, verinden güçlü özetler çıkarırsın.

Beş temel fonksiyon

- **COUNT:** satır/değer sayısı.
- **SUM:** sayısal toplam.
- **AVG:** ortalama.
- **MAX / MIN:** en büyük / en küçük.



Şema 16.1 — SQL toplama fonksiyonları: bir grubu tek değere indirir.

Toplama fonksiyonları

```
 -- Her kategorinin toplam ve ortalama fiyatı:
SELECT kategori,
       COUNT(*)   AS urun_sayisi,
       SUM(fiyat) AS toplam,
       AVG(fiyat) AS ortalama
FROM urunler
GROUP BY kategori;
```

İPUCU

`COUNT(*)` tüm satırları sayar; `COUNT(sutun)` ise o sütunda NULL olmayan değerleri sayar — bu fark bazen önemlidir. Toplama fonksiyonları GROUP BY olmadan da kullanılabilir; o zaman tüm tabloyu tek bir gruba indirirler (örn. `SELECT AVG(fiyat) FROM urunler` tüm ürünlerin ortalama fiyatını verir). Bu fonksiyonlar, raporlama ve analizin temelidir.

Sorgu ne döndürür? SONUÇ

Bu sorgu her kategori için üç özet üretir: ürün sayısı (COUNT), fiyatların toplamı (SUM) ve ortalaması (AVG). GROUP BY ile birleştiğinde, "hangi kategori en çok ürüne / en yüksek ortalamaya sahip?" gibi sorulara tek sorguda yanıt alırsın.

Alıştırma 12 dk

Özet çıkar:

1. Tüm siparişlerin toplam tutarını bulan sorguyu yaz (SUM).
2. Her kullanıcının ortalama sipariş tutarını GROUP BY ile bul.
3. COUNT(*) ile COUNT(sutun) arasındaki farkı açıkla.

BÖLÜM 17

Grupları Süzmek: HAVING

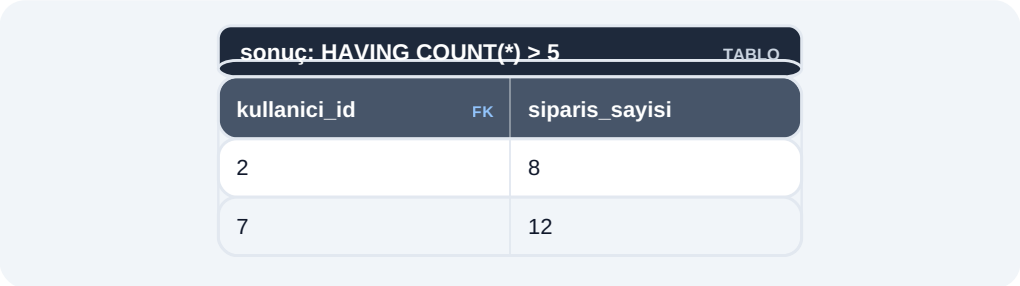
WHERE, satırları süzer; ama gruplandıktan sonra grupları süzmek istersen (örn. "yalnızca 5'ten fazla siparişi olan kullanıcılar") HAVING kullanılır. HAVING, GROUP BY'ın sonucuna uygulanan bir filtredir.

WHERE ve HAVING farkı

- **WHERE:** gruplamadan **önce**, tek tek satırları süzer.
- **HAVING:** gruplamadan **sonra**, grupları süzer.
- HAVING genelde bir toplama fonksiyonu içerir (örn. COUNT(*) > 5).

HAVING ile grup süzme

```
 -- 5'ten fazla siparişi olan kullanıcılar:
SELECT kullanici_id, COUNT(*) AS siparis_sayisi
FROM siparisler
GROUP BY kullanici_id
HAVING COUNT(*) > 5;
```



Şema 17.1 — HAVING, gruplar arasından yalnızca koşulu sağlayanları bırakır.

İPUCU

Anahtar ayrım: **WHERE satırlara, HAVING gruplara** uygulanır. "Fiyatı 100'den büyük ürünleri kategorilere göre say" derken: önce `WHERE fiyat > 100` (satırları süz), sonra `GROUP BY kategori`, sonra istersen `HAVING COUNT(*) > 3` (grupları süz). İkisi aynı sorguda birlikte kullanılabilir. Toplama sonucuna göre süzmek istiyorsan WHERE değil HAVING gerekir.

Sorgu ne döndürür? SONUÇ

Bu sorgu önce siparişleri kullanıcıya göre gruplar, her grubun sayısını bulur, sonra `HAVING` ile yalnızca 5'ten fazla siparişi olan grupları bırakır. Sonuç: "çok sipariş veren" kullanıcıların listesi — sadık müşterileri bulmanın bir yolu.

Alıştırma

12 dk

HAVING yaz:

1. "10'dan fazla ürünü olan kategoriler" sorgusunu yaz.
2. WHERE ile HAVING arasındaki farkı bir örnekle açıkla.
3. Aynı sorguda hem WHERE hem HAVING kullanılan bir durum düşün.

BÖLÜM 18

Normalizasyon

Normalizasyon, veriyi tekrar etmeyecek ve tutarsızlığa düşmeyecek biçimde tablolara bölmektir. Tek bir şişman tabloda her şeyi tutmak ilk başta kolay görünür ama veri tekrarına ve hatalara yol açar.

Neden bölmeli?

- Veri tekrarı yer israfı ve tutarsızlık riskidir.
- Bir bilgi tek bir yerde tutulmalı (örn. kullanıcı adı).
- Tekrar eden veriyi ayrı tabloya çıkar, ilişkiyle bağla.

Şema 18.1 — Normalizasyon: tekrarı önlemek için veriyi ilişkili tablolara böl.

İPUCU

Klasik örnek: siparişler tablosunda her satıra kullanıcının adını, adresini, telefonunu yazarsan, aynı kullanıcının her siparişinde bu bilgiler tekrarlanır. Kullanıcı adresini değiştirince **tüm** siparişlerini güncellemen gerekir — birini atlarsan veri tutarsız olur. Çözüm: kullanıcı bilgisini bir kez `kullanicilar` tablosunda tut, siparişlerde sadece `kullanici_id` (FK) bulundur. "Her gerçek tek bir yerde" — normalizasyonun özü budur.

Sorgu ne döndürür? SONUÇ

Normalize edilmiş tasarımda kullanıcı adı yalnızca `kullanicilar` tablosunda bir kez bulunur; siparişler ona `kullanici_id` ile bağlanır. Ad değişince tek bir satırı güncellersin ve tüm siparişler otomatik olarak doğru adı (JOIN ile) gösterir. Tekrar yok, tutarsızlık yok.

Alıştırma

12 dk

Normalize et:

1. Her satırda müşteri bilgisini tekrar eden bir "şişman" tablo düşün.
2. Bu tabloyu iki ilişkili tabloya böl.
3. Bölmenin hangi tutarsızlık riskini ortadan kaldırdığını açıkla.

BÖLÜM 19

İndeksler (Index)

Bir tablo büyüdükçe, bir değeri aramak yavaşlar — çünkü veritabanı tüm satırları taramak zorunda kalabilir. İndeks, tıpkı bir kitabın dizini gibi, aranan değere doğrudan gitmeyi sağlayarak sorguları çok hızlandırır.

İndeks ne yapar?

- İndekssiz arama: tüm satırları tek tek tara (yavaş).
- İndeksli arama: doğrudan ilgili satıra atla (hızlı).
- Sık aranan/filtrelenen sütunlara indeks eklenir.

Şema 19.1 — İndeks, aramayı "hepsini tara"dan "doğrudan bul"a çevirir.

İPUCU

İndeksin bir bedeli vardır: aramayı hızlandırır ama **yazma işlemlerini (INSERT/UPDATE) biraz yavaşlatır** ve yer kaplar (dizin de güncellenmeli). Bu yüzden her sütuna değil, **sık aranan/filtrelenen/JOIN edilen** sütunlara indeks koyarsın. Birincil anahtarlar genelde otomatik indekslenir. Kural: önce sorgu yaz, yavaşsa ve sık çalışıyorsa indeks ekle — erken ve aşırı indeksleme ters teper.

Sorgu ne döndürür? SONUÇ

İndekssiz bir tabloda `WHERE eposta = 'x@y.com'` milyonlarca satırı taramak zorunda kalabilir. `eposta` sütununa indeks eklersen, veritabanı dizini kullanıp doğru satıra neredeyse anında ulaşır — sorgu saniyelerden milisaniyelere iner.

Alıştırma

10 dk

İndeksi düşün:

1. Bir "kullanicilar" tablosunda hangi sütuna indeks eklerdin? Neden?
2. İndeksin yazma işlemlerine etkisini açıkla.
3. Neden her sütuna indeks eklemek iyi fikir değildir?

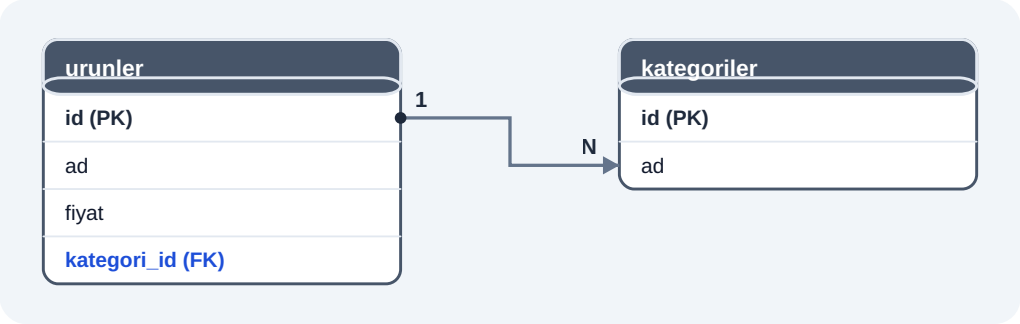
BÖLÜM 20

Veritabanı Tasarımı

İyi bir veritabanı, kod yazmadan önce iyi tasarlanır. Hangi tablolar, hangi sütunlar, hangi ilişkiler? Bu kararları baştan doğru vermek, ileride çok büyük yeniden yazımları önler.

Tasarım adımları

- **Varlıkları belirle:** sistemdeki "şeyler" (kullanıcı, ürün, sipariş).
- **Her varlık bir tablo:** sütunlarını ve birincil anahtarını tanımla.
- **İlişkileri kur:** yabancı anahtarlarla bağla.
- **Normalize et:** tekrarı önle.



Şema 20.1 — Tasarım örneği: ürünler bir kategoriye bağlanır (N ürün → 1 kategori).

İPUCU

Tasarıma **kâğıt üstünde bir ER (Varlık-İlişki) şemasıyla** başla: kutular tablolar, oklar ilişkiler. Kodlamadan önce şemayı netleştirmek, Modül 6'daki "önce planla, sonra kodla" ilkesinin veritabanı karşılığıdır. İyi sorular: "Bu varlık başka neye bağlı? Bir X'in kaç Y'si olur (bir-bir, bir-çok, çok-çok)? Hangi bilgi nerede, tek bir yerde tutuluyor mu?" Çok-çok ilişkiler genelde bir ara tablo gerektirir.

Sorgu ne döndürür? SONUÇ

Bu tasarımda her ürün bir `kategori_id` ile bir kategoriye bağlıdır; bir kategorinin birçok ürünü olabilir (çok-bir). Kategori adı yalnızca `kategoriler` tablosunda bir kez tutulur. İyi tasarlanmış bir şema, sorguları kolay, veriyi tutarlı ve sistemi büyümeye hazır kılar.

Alıştırma

16 dk

Şema tasarla:

1. Bir blog için varlıkları belirle (yazılar, yazarlar, yorumlar, etiketler).
2. Her varlık için bir tablo ve birincil anahtar tasarla.
3. Tablolar arası ilişkileri (yabancı anahtarlarla) çiz.

SEVİYE 4

Güvenlik ve Üretim

Gerçek dünya: SQL enjeksiyonu ve parametreli sorgu, işlemler (transactions), görünümler, yedekleme, NoSQL'e bakış ve küçük bir şema tasarlamak.

BÖLÜM 21

SQL Enjeksiyonu ve Parametreli Sorgu

SQL enjeksiyonu, en yaygın ve en tehlikeli güvenlik açıklarından biridir: kullanıcı girdisi doğrudan bir SQL sorgusuna gömülürse, saldırgan kendi SQL'ini çalıştırabilir. Çözüm basit ve kesindir: parametreli sorgular.

Tehlike ve çözüm

- **Tehlike:** kullanıcı girdisini metin olarak sorguya yapıştırmak.
- **Sonuç:** saldırgan sorguyu değiştirip veri çalabilir/silebilir.
- **Çözüm:** parametreli sorgu — girdi "veri" olarak işlenir, "kod" olarak değil.

Şema 21.1 — Girdiyi birleştirmek tehlikelidir; parametreli sorgu güvenlidir.

Tehlikeli vs güvenli (kavramsal)

```
// TEHLİKELİ: girdi doğrudan sorguya gömülür
sorgu = "SELECT * FROM kullanicilar WHERE ad = '" + girdi + "'"

// GÜVENLİ: parametreli sorgu (yer tutucu kullan)
sorgu = "SELECT * FROM kullanicilar WHERE ad = ?"
calistir(sorgu, [girdi])   // girdi yalnızca veri olarak gider
```

İPUCU

Bu, Backend modülündeki "kullanıcı verisine asla güvenme" ilkesinin somut hâlidir. Kural mutlaktır: **kullanıcı girdisini asla SQL metnine birleştirme**; her zaman parametreli sorgu (hazırlanmış ifadeler / prepared statements) kullan. Parametreli sorguda girdi, sorgunun yapısını değiştiremez — sadece bir değer olarak yerine konur. Bu tek alışkanlık, SQL enjeksiyonunu neredeyse tamamen ortadan kaldırır. PHP, Python ve C# modüllerinde bunun dile özgü biçimlerini göreceksin.

Sorgu ne döndürür? SONUÇ

Güvenli sürümde `?` bir yer tutucudur; veritabanı önce sorgunun yapısını derler, sonra girdiyi yalnızca bir **değer** olarak yerleştirir. Böylece saldırgan girdiye SQL kodu yazsa bile, o kod çalışmaz — sadece aranan bir metin olarak ele alınır. Sorgu yapısı saldırgana kapalıdır.

Alıştırma 12 dk

Güvenliği uygula:

1. Girdiyi birleştiren tehlikeli bir sorguyu parametreli hâle çevir.
2. Parametreli sorgunun enjeksiyonu neden engellediğini açıkla.
3. "Kullanıcı verisine güvenme" ilkesini SQL bağlamında yaz.

BÖLÜM 22

İşlemler (Transactions)

Bazı işlemler birden çok adımdan oluşur ve hepsi ya birlikte başarılı olmalı ya da hiçbiri olmamalı. Para transferi gibi: bir hesaptan düş, diğerine ekle. İşlemler (transactions), bu "ya hep ya hiç" güvencesini sağlar.

İşlem mantığı

- `BEGIN` ile başla, adımları çalıştır.
- Hepsi başarılıysa `COMMIT` (kalıcı yap).
- Bir adım başarısızsa `ROLLBACK` (hepsini geri al).

Şema 22.1 — İşlem: tüm adımlar birlikte kalıcı olur ya da hiçbiri olmaz.

İPUCU

İşlemler veri bütünlüğünün koruyucusudur. Para transferinde ilk UPDATE çalışıp ikincisi (örn. elektrik kesintisi) çalışmazsa, para "buharlaşır". İşlem içine alırsan, ikinci adım başarısız olunca ilk adım da geri alınır (ROLLBACK) — hesaplar tutarlı kalır. Bu güvenceye **ACID** denir (Atomiklik, Tutarlılık, İzolasyon, Dayanıklılık). Kritik, çok adımlı işlemleri her zaman bir işlem içinde yap.

Sorgu ne döndürür? SONUÇ

Bir işlem içinde iki UPDATE çalıştırırsın; ikisi de başarılıysa `COMMIT` değişiklikleri kalıcı yapar. İkincisi başarısız olursa `ROLLBACK` ilk değişikliği de geri alır — sanki hiçbir şey olmamış gibi. Sonuç: veritabanı asla "yarım" bir durumda kalmaz.

Alıştırma 12 dk

İşlem tasarla:

1. Para transferinin neden tek bir işlem olması gerektiğini açıkla.
2. Bir adım başarısız olursa ne yapılmalı (COMMIT mi ROLLBACK mı)?
3. Çok adımlı bir işlemi BEGIN/COMMIT/ROLLBACK ile sözde-yaz.

BÖLÜM 23

Görünümler (Views)

Sık kullanılan karmaşık bir sorguyu her seferinde yeniden yazmak yerine, onu bir görünüm (view) olarak kaydedebilirsin. Görünüm, kayıtlı bir sorgudur; ona basit bir tablo gibi SELECT atarsın.

Görünüm ne işe yarar?

- Karmaşık bir sorguyu bir isimle "paketler".
- Tekrar tekrar yazmak yerine basitçe `SELECT * FROM view`.
- Kullanıcılardan karmaşıklığı ve bazı sütunları gizleyebilir.

Şema 23.1 — Görünüm: karmaşık bir sorguyu basit bir isim ardına gizler.

İPUCU

Görünümler, Modül 6'daki **soyutlama** fikrinin veritabanı karşılığıdır: karmaşık mantığı bir kez yaz, anlamlı bir isim ver, sonra o ismi defalarca kullan. Ek bir fayda: bir görünüm yalnızca belirli sütunları gösterecek şekilde tanımlanabilir, böylece bazı kullanıcılara hassas sütunları (örn. maaş) göstermeden veriye erişim verebilirsin. Görünümler genelde veriyi kopyalamaz; her erişimde alttaki sorgu yeniden çalışır.

Sorgu ne döndürür? SONUÇ

`aktif_musteriler` görünümü, arkada karmaşık bir JOIN ve filtre sorgusu çalıştırır; ama sen ona `SELECT * FROM aktif_musteriler` diyerek basit bir tablo gibi erişirsin. Karmaşıklık bir kez yazılır ve isim ardına gizlenir — kod hem kısalır hem okunur.

Alıştırma

10 dk

Görünüm düşün:

1. Hangi karmaşık sorgunu bir görünüm yapardın? Neden?
2. Görünümün soyutlama ile ilişkisini açıkla.
3. Görünümün hassas sütunları gizlemede nasıl yardımcı olduğunu yaz.

BÖLÜM 24

Yedekleme ve Kurtarma

Veri, çoğu sistemin en değerli varlığıdır — ve donanım arızası, hatalı bir DELETE veya saldırı bir anda yok edebilir. Düzenli yedekleme, "felaket" anında veriyi geri getirmenin tek güvencesidir.

Yedekleme ilkeleri

- **Düzenli ve otomatik:** elle değil, zamanlanmış yedek.
- **Birden çok kopya:** farklı yerlerde (yerel + uzak).
- **Geri yüklemeyi test et:** denenmemiş yedek, yedek sayılmaz.

Şema 24.1 — Yedekleme: felaketten önce alınır, felaketten sonra hayat kurtarır.

İPUCU

Altın kural **3-2-1**: en az **3** kopya, **2** farklı ortamda, **1**'i fiziksel olarak başka bir yerde (uzak/bulut). En sık yapılan hata, yedek almak ama **geri yüklemeyi hiç test etmemektir** — bozuk veya eksik bir yedeğin değeri sıfırdır. Bölüm 10-11'deki WHERE'siz UPDATE/DELETE felaketlerinden kurtulmanın son çaresi de iyi bir yedektir. Yedekleme bir maliyet değil, sigortadır.

Sorgu ne döndürür? SONUÇ

İyi bir yedekleme düzeninde, veritabanı düzenli aralıklarla otomatik olarak kopyalanır ve kopyalar farklı yerlerde saklanır. Bir felaket anında (örn. yanlışlıkla silinen tablo), en son sağlam yedekten geri yükleyerek veriyi kurtarırsın — yeter ki o yedeğin çalıştığını önceden test etmiş ol.

Alıştırma

8 dk

Yedekleme planla:

1. Bir uygulama için yedekleme sıklığını ve yerlerini belirle.
2. 3-2-1 kuralını kendi planında uygula.
3. "Geri yüklemeyi test etmek" neden bu kadar önemli, açıkla.

BÖLÜM 25

NoSQL'e Kısa Bakış

İlişkisel veritabanları (SQL) çoğu iş için harikadır; ama bazı durumlar farklı yaklaşımlar gerektirir. NoSQL veritabanları, esneklik ve ölçek için tabloların yerine farklı veri modelleri (belge, anahtar-değer) kullanır.

SQL ve NoSQL

- **SQL (ilişkisel):** tablolar, katı şema, güçlü ilişkiler ve JOIN.
- **NoSQL (belge/anahtar-değer):** esnek yapı, kolay ölçek, JOIN az.
- Çoğu projede SQL yeterli ve daha güvenlidir; NoSQL belirli ihtiyaçlar için.

Şema 25.1 — SQL ve NoSQL: iki farklı veri modeli felsefesi.

İPUCU

"NoSQL daha yeni, o yüzden daha iyi" düşüncesi yaygın bir yanılgıdır. İkisi de güçlüdür ama farklı işler için: katı ilişkiler ve tutarlılık gerektiren çoğu iş uygulaması (e-ticaret, finans, kayıt sistemleri) için **SQL hâlâ ilk tercihtir**. NoSQL ise çok büyük ölçek, esnek/değişken şema veya basit anahtar-değer erişimi gerektiren senaryolarda parlar. Seçim "moda"ya değil, **probleme** göre yapılır — Backend modülündeki mimari dersiyle aynı ilke.

Sorgu ne döndürür? SONUÇ

SQL'de veri katı tablolara ve ilişkilere göre düzenlenir; NoSQL'de ise (örneğin belge tabanlı bir veritabanında) bir kayıt, JSON benzeri esnek bir belge olabilir. İlişkiler ve tutarlılık önemliyse SQL, esneklik ve büyük ölçek önemliyse NoSQL öne çıkar — ama bu modülde öğrendiğin temeller her iki dünyada da işine yarar.

Alıştırma

10 dk

Karşılaştır:

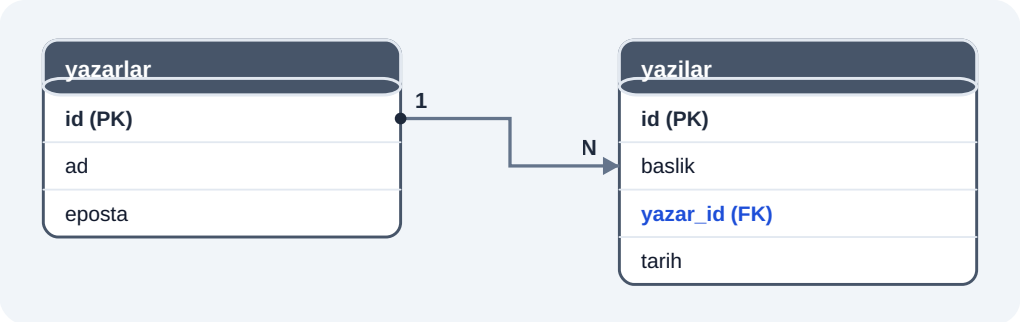
1. Bir bankacılık sistemi için SQL mi NoSQL mi? Gerekçeni yaz.
2. NoSQL'in bir avantajını ve bir dezavantajını söyle.
3. Seçimin "moda" değil "problem" temelli olması gerektiğini açıkla.

BÖLÜM 26

Bitirme: Küçük Bir Şema Tasarlamak

Tüm öğrendiklerini birleştirip baştan sona küçük bir veritabanı tasarlıyorsun: tablolar, anahtarlar, ilişkiler ve birkaç temel sorgu. Bu, gerçek bir projenin veri katmanının çekirdeğidir.

Örnek: basit bir blog şeması



Şema 26.1 — Blog şeması: bir yazarın çok yazısı olur (1 → N).

Şema + örnek sorgular

```
 -- Tasarım kontrol listesi:
-- 1. Varlıklar: yazarlar, yazilar  (her biri bir tablo)
-- 2. Birincil anahtarlar: id (PK)
-- 3. İlişki: yazilar.yazar_id -> yazarlar.id (FK)
-- 4. Normalize: yazar bilgisi tek yerde

-- Her yazıyı yazar adıyla getir:
SELECT yazilar.baslik, yazarlar.ad
FROM yazilar
INNER JOIN yazarlar ON yazilar.yazar_id = yazarlar.id;
```

İPUCU

Gerçek bir şema tasarlarken bu sırayı izle: varlıkları belirle → her birini tablo yap → birincil anahtarları koy → ilişkileri yabancı anahtarlarla kur → normalize et (tekrarı önle) → sık aranan sütunlara indeks ekle → güvenliği (parametreli sorgular) baştan düşün. Bu modülü tamamladıysan, artık verinin nasıl saklandığını ve sorgulandığını biliyorsun. Sıradaki adım, bu veritabanına gerçek bir dille (PHP, Python, C#) bağlanıp uygulamanı hayata geçirmek.

Sorgu ne döndürür? SONUÇ

Tasarladığın blog şemasında her yazı bir `yazar_id` ile bir yazara bağlıdır. Yukarıdaki JOIN, her yazıyı yazarının adıyla birlikte tek bir sonuçta döner. Tablolar, anahtarlar, ilişki ve sorgu bir araya gelince — çalışan, tutarlı ve büyümeye hazır bir veri katmanın olur.

Alıştırma 20 dk

Şemanı tasarla:

1. Bir "etkinlik kayıt" sistemi seç; varlıklarını belirle.
2. Tabloları, birincil ve yabancı anahtarlarıyla tasarla.
3. İki tabloyu birleştiren bir JOIN sorgusu yaz.
4. Güvenlik (parametreli sorgu) ve bir indeks kararını tasarıma ekle.

EK

SQL & Veritabanı Terimleri Sözlüğü

En sık kullanılan SQL komutları ve veritabanı terimleri. Bir başvuru kaynağı olarak saklayabilirsin.

SELECT	`Veri oku`	FROM	`Hangi tablo`
WHERE	`Satırları süz`	ORDER BY	`Sırala`
LIMIT	`Sayıyı sınırla`	DISTINCT	`Benzersiz değerler`
INSERT	`Satır ekle`	UPDATE	`Satır güncelle`
DELETE	`Satır sil`	JOIN	`Tabloları birleştir`
GROUP BY	`Grupla + özetle`	Primary Key	`Benzersiz kimlik`
Foreign Key	`İlişki anahtarı`	Index	`Hızlı arama`

SQL'in özeti SONUÇ

SQL ile veriyi dört temel işlemle yönetirsin: **SELECT** (oku), **INSERT** (ekle), **UPDATE** (güncelle), **DELETE** (sil). Tabloları **birincil anahtar** ile tanımlar, **yabancı anahtar** ve **JOIN** ile ilişkilendirir, **GROUP BY** ile özetlersin. Güvenliğin altın kuralı: kullanıcı verisini sorguya gömme, her zaman **parametreli sorgu** kullan — böylece SQL enjeksiyonundan korunursun.