

Bölüm 10: Veri Tabanı

Java ile Nesne Yönelimli Programlama

Kalıcı depolama

İlişkisel model

SQLite

SQL komutları

JDBC

Ekle, oku, güncelle, sil

PreparedStatement

İşlemler (transaction)

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

Öğrenme Çıktıları

Kavramlar

Veri tabanının neden gerektiğini, dosyalardan farkını ve ilişkisel modeli (tablo, satır, sütun, birincil anahtar) açıklamak.

SQL

`CREATE TABLE` , `INSERT` , `SELECT` , `UPDATE` , `DELETE` komutlarını yazmak; `WHERE` , `ORDER BY` , `LIMIT` ile sorguyu daraltmak.

JDBC

`Connection` , `Statement` ve `ResultSet` ile Java'dan SQLite'a bağlanıp kayıt eklemek, okumak, güncellemek ve silmek; kaynakları try-with-resources ile kapatmak.

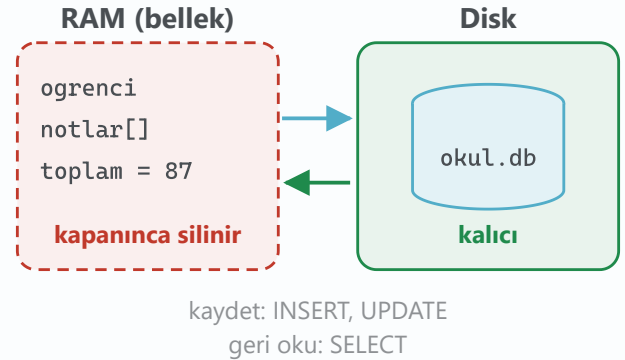
Güvenlik ve bütünlük

SQL enjeksiyonunu

`PreparedStatement` ile önlemek; `commit` / `rollback` ile birden çok komutu tek bir işlem olarak yürütmek.

Veri Tabanı Neden Önemlidir?

- Programdaki değişkenler ve nesneler **RAM**'de yaşar: program kapanınca **kaybolur**.
- **Kalıcı depolama** (persistent storage): verinin uzun süre, program kapandıktan sonra da saklanması. Öğrenci notları, banka hesapları, oyun skorları hep kalıcı olmalıdır.
- **Veri tabanı** (database): düzenli biçimde saklanan ve kolayca sorgulanabilen veri topluluğu. Onu yöneten yazılıma **Veri tabanı Yönetim Sistemi** (VTYS, DBMS) denir.
- Benzetme: veri tabanı, kütüphanenin **kataloglu arşivi** gibidir; aradığınız kitabı tüm rafları gezmeden bulursunuz.



Program yeniden açılınca veri diskten okunur.

Dosya mı, Veri Tabanı mı?

Dosya sistemleri bazı uygulamalar için **yeterli** olabilir (ayar dosyası, günlük kayıt, kısa bir liste). Veri büyüyüp karmaşılaştıkça **veritabanları gerekir**: verinin verimli işlenmesi için önemli özellikler sunarlar.

Özellik	Düz dosya (txt, csv)	Veri tabanı
Arama	Dosya baştan sona okunur	İndeksle hızlı; tek satırlık SQL sorgusu
Güncelleme	Çoğu zaman tüm dosya yeniden yazılır	Yalnızca ilgili satır değişir
Eş zamanlı erişim	İki program aynı anda yazarsa veri bozulabilir	Kilitleme ve işlemlerle güvenli
Tutarlılık	Biçim denetimi tamamen programcıya kalır	Veri tipleri ve kısıtlar (NOT NULL , UNIQUE , CHECK)
Güvenlik	Dosya izni: ya hepsi ya hiçbiri	Kullanıcı, tablo ve alan düzeyinde yetki
Hata sonrası	Yarım kalan yazma dosyayı bozabilir	Yarım kalan işlem geri alınır



Temel Özellikler

Verilere hızlı erişim

İndeksler sayesinde milyonlarca satır arasından aranan kayıt kısa sürede bulunur.

Alan seviyesinde güvenlik

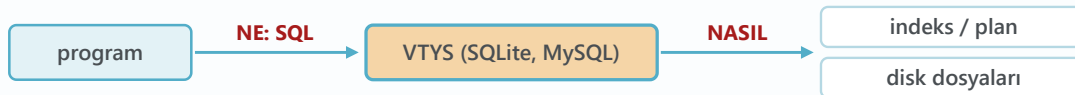
Yetki tablo, hatta sütun (alan) düzeyinde verilebilir: maaş sütununu yalnızca muhasebe birimi görebilir.

Platform bağımsız, standart API'ler

SQL bir standarttır; Java'da **JDBC** aynı arayüzle farklı veritabanlarına erişir.

Dosya yapısından bağımsızlık

Program verinin diskte nasıl tutulduğunu bilmez. **Ne** istediğini SQL ile söyler; **nasıl** bulunacağına VTYS karar verir.



Bunlara ek olarak veritabanları veri bütünlüğü kısıtları ve işlem (transaction) desteği sunar; bölümün sonunda göreceğiz.

İlişkisel Model: Tablo, Satır, Sütun

users tablo (table)

sütun (alan, column)

satır (kayıt, row)

id	name	email
1	Ahmet Yılmaz	ahmet@example.com
2	Ayşe Demir	ayse@example.com
3	Can Kaya	can@example.com

birincil anahtar (primary key): her satırda farklı

- **İlişkisel** (relational) veri tabanı veriyi **tablolarda** saklar.
- **Sütun**: verinin bir özelliği; adı ve veri tipi vardır (`name TEXT`).
- **Satır**: bir varlığın tüm bilgileri (bir kullanıcı).
- **Birincil anahtar**: her satırı benzersiz tanımlar; iki satırın `id` 'si aynı olamaz.
- Java benzetmesi: tablo $\approx$ **sınıf**, sütun $\approx$ **alan**, satır $\approx$ **nesne**.

Veri Tabanı Türleri

İlişkisel

Tablolar ve SQL. **Oracle, SQL Server, MySQL, PostgreSQL**. Kullanımın büyük bölümü (yaklaşık %77).

NoSQL / arama

Belge, anahtar-değer, arama dizini gibi tablosuz modeller; belirli senaryolarda kullanılır: **MongoDB, Elasticsearch** (yaklaşık %23).

Gömülü ilişkisel

SQLite: programın içinde çalışan ilişkisel veri tabanı; veri paylaşımının az olduğu durumlarda yaygındır.

İlişkisel



%77

NoSQL / arama



%23

yaklaşık kullanım payı

Bu bölümde, kurulumu en kolay olan SQLite'ı kullanacağız. Öğrendiğiniz SQL ve JDBC kodu diğer ilişkisel veritabanlarında da neredeyse aynıdır.

SQLite Nedir?

- **Gömülü** (embedded) bir ilişkisel veri tabanıdır: programın içinde bir **kütüphane** olarak çalışır.
- **Sunucuya ihtiyaç duymaz** (serverless): ayrı bir program, kurulum ya da yönetici gerekmez.
- **Kendine yeten** bir yapıya sahiptir: tüm tablolar **tek bir disk dosyasında** saklanır (ör. `sample.db`).
- **Çapraz platform**: aynı dosya Windows, Linux, macOS ve Android'de açılır.
- Kolayca kurulur ve kullanılır; **açık kaynaklı ve ücretsizdir**.
- Dünyada en yaygın kullanılan gömülü ilişkisel veri tabanıdır.

Yaygın kullanım alanları

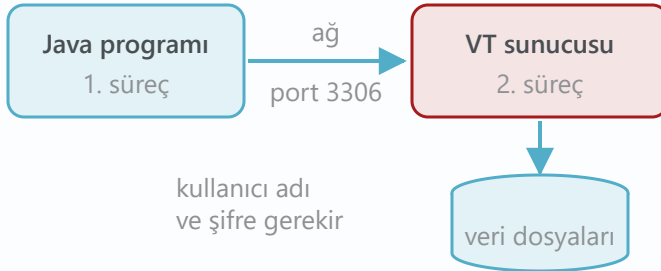
- + Mobil uygulamalar
- + Gömülü sistemler
- + Küçük ölçekli uygulamalar
- + Tarayıcı, e-posta istemcisi gibi masaüstü programları

Uygun olmadığı yer

- Ağ üzerinden çok sayıda kullanıcının aynı anda yazdığı büyük sistemler

Sunucu Tabanlı ve Gömülü Veri Tabanı

Sunucu tabanlı (MySQL, Oracle)



Gömülü (SQLite)



	Sunucu tabanlı	SQLite
Kurulum	Sunucu kurulur, kullanıcı ve şifre tanımlanır	Projeye tek bir jar eklenir
Veri	Sunucunun yönettiği birçok dosya	Tek bir <code>.db</code> dosyası
Eş zamanlı yazma	Çok kullanıcı	Aynı anda tek yazıcı

SQLite Sürücüsü ve DB Browser

SQLite JDBC sürücüsü

Java'nın SQLite ile konuşmasını sağlayan jar dosyası (`sqlite-jdbc-3.x.jar`):
github.com/xerial/sqlite-jdbc

DB Browser for SQLite

SQLite için popüler bir SQL istemcisi: `.db` dosyasını açar, tabloları görsel gösterir, SQL çalıştırır: sqlitebrowser.org/dl

Eclipse'te sürücüyü projeye eklemek

1. Jar dosyasını indirip projede bir `lib` klasörüne koyun.
2. Projeye sağ tık → **Build Path** → **Configure Build Path...**
3. **Libraries** sekmesi → **Classpath** → **Add JARs...** → jar'ı seçin → **Apply and Close**.

Komut satırından

```
javac Baglan.java  
java -cp ".;sqlite-jdbc-3.40.0.0.jar" Baglan
```

Linux ve macOS'ta ayırıcı `;` yerine `:` olur. Derleme için sürücü gerekmez;
`java.sql` paketi JDK ile gelir.

Yaygın SQL Komutları

Komut	Görevi	Örnek	CRUD
CREATE TABLE	Yeni tablo oluşturur	CREATE TABLE users (id INTEGER, name TEXT)	—
INSERT INTO	Yeni kayıt ekler	INSERT INTO users(name) VALUES('Ali')	Create
SELECT	Tabloyu sorgular	SELECT * FROM users	Read
UPDATE	Var olan kayıtları günceller	UPDATE users SET name = 'Efe' WHERE id = 1	Update
DELETE	Kayıtları siler	DELETE FROM users WHERE id = 1	Delete



- **SQL** (Structured Query Language, yapılandırılmış sorgu dili): ilişkisel veritabanlarının ortak dilidir.
- Anahtar kelimelerde büyük/küçük harf fark etmez; okunabilirlik için BÜYÜK yazılır. Metin değerler **tek tırnak** içindedir: 'Ali'.
- * "tüm sütunlar" demektir. WHERE komutun hangi satırlara uygulanacağını seçer.

CREATE TABLE ve Veri Tipleri

```
CREATE TABLE IF NOT EXISTS uyeler (  
  id    INTEGER PRIMARY KEY,  -- benzersiz  
  name  TEXT NOT NULL,        -- boş olamaz  
  email TEXT UNIQUE,          -- tekrar edemez  
  yas   INTEGER CHECK (yas >= 0)  
)
```

- `IF NOT EXISTS` : tablo zaten varsa hata verme.
- `INTEGER PRIMARY KEY` : değer verilmezse SQLite **en büyük id + 1** atar.
- Kısıt bozulursa komut reddedilir: `NOT NULL constraint failed: uyeler.name`, `CHECK constraint failed: yas >= 0`.

SQLite tipi	Java karşılığı
INTEGER	<code>int</code> , <code>long</code> · <code>getInt</code>
REAL	<code>double</code> · <code>getDouble</code>
TEXT	<code>String</code> · <code>getString</code>
BLOB	<code>byte[]</code> · <code>getBytes</code>
NULL	değer yok

SQL Komutları Adım Adım

```
CREATE TABLE users (id INTEGER PRIMARY KEY,  
                      name TEXT, email TEXT)
```

users

id	name	email
(tablo boş)		

Tablo oluşturuldu: üç sütun var, henüz **hiç satır yok**.

SQL Komutları Adım Adım

```
INSERT INTO users(name, email)
VALUES('Ahmet Yılmaz', 'ahmet@example.com')
```

users

id	name	email
1	Ahmet Yılmaz	ahmet@example.com

id verilmedi; SQLite **1** atadı. 1 satır etkilendi.

SQL Komutları Adım Adım

```
INSERT INTO users(name, email)
VALUES('Ayşe Demir', 'ayse@example.com')
```

users

id	name	email
1	Ahmet Yılmaz	ahmet@example.com
2	Ayşe Demir	ayse@example.com

İkinci kayıt `id = 2` aldı (en büyük id + 1).

SQL Komutları Adım Adım

```
UPDATE users SET name = 'Mehmet Akdeniz'  
WHERE id = 1
```

users

id	name	email
1	Mehmet Akdeniz	ahmet@example.com
2	Ayşe Demir	ayse@example.com

`WHERE id = 1` yalnızca ilk satırı seçti; sadece `name` değişti, `email` aynı kaldı.

SQL Komutları Adım Adım

```
DELETE FROM users WHERE id = 1
```

users

id	name	email
1	Mehmet Akdeniz	ahmet@example.com
2	Ayşe Demir	ayse@example.com

id = 1 olan satır silindi. 1 satır etkilendi.

SQL Komutları Adım Adım

```
SELECT * FROM users
```

users

id	name	email
2	Ayşe Demir	ayse@example.com

Sorgu tabloyu **değiştirmez**, yalnızca okur: sonuç tek satır (2, Ayşe Demir, ayse@example.com) .

SELECT: Süzme, Sıralama, Sınırlama

skorlar

oyuncu	puan
ada	72
can	95
ece	58
efe	95
naz	81

Sorgu	Sonuç
SELECT oyuncu, puan FROM skorlar WHERE puan < 90 ORDER BY puan DESC	naz 81, ada 72, ece 58
SELECT oyuncu, puan FROM skorlar ORDER BY puan DESC, oyuncu LIMIT 3	can 95, efe 95, naz 81
SELECT oyuncu, puan FROM skorlar WHERE oyuncu LIKE 'e%'	ece 58, efe 95
SELECT COUNT(*), AVG(puan), MAX(puan) FROM skorlar WHERE puan >= 60	4 · 85.75 · 95

- WHERE satırları süzer (= , <> , < , >= , AND , OR , LIKE 'e%' : "e ile başlayan").
- ORDER BY ... DESC büyükten küçüğe sıralar; eşitlikte ikinci ölçüt (oyuncu) kullanılır. LIMIT 3 ilk 3 satırı alır.

Bir **SELECT** Sorgusu Nasıl Çalışır?

```
▶ String sql = "SELECT oyuncu, puan FROM skorlar "  
▶       + "WHERE puan > 70 "  
▶       + "ORDER BY oyuncu DESC LIMIT 2";  
4 try (Statement st = conn.createStatement();  
5     ResultSet rs = st.executeQuery(sql)) {  
6     while (rs.next()) {  
7         System.out.println(rs.getString("oyuncu") + " "  
8                               + rs.getInt("puan"));  
9     }  
10 }
```

FROM

WHERE

ORDER BY

LIMIT

oyuncu	puan
ada	72
can	95
ece	58
efe	95
naz	81

Sorgu tek bir metindir. **skorlar** tablosunda 5 satır var.

Adım 1 / 10

DEĞİŞKENLER

sql = **SELECT ... LIMIT 2**

ÇIKTI



Bir **SELECT** Sorgusu Nasıl Çalışır?

```
1 String sql = "SELECT oyuncu, puan FROM skorlar "  
2     + "WHERE puan > 70 "  
3     + "ORDER BY oyuncu DESC LIMIT 2";  
4 try (Statement st = conn.createStatement();  
5     ▶ ResultSet rs = st.executeQuery(sql)) {  
6     while (rs.next()) {  
7         System.out.println(rs.getString("oyuncu") + " "  
8             + rs.getInt("puan"));  
9     }  
10 }
```

FROM**WHERE****ORDER BY****LIMIT**

oyuncu	puan
ada	72
can	95
ece	58
efe	95
naz	81

5 satır

executeQuery sorguyu gönderir. Veri tabanı önce **FROM** : tablonun bütün satırları.

Adım 2 / 10

DEĞİŞKENLER

aşama = FROM

ÇIKTI



Bir **SELECT** Sorgusu Nasıl Çalışır?

```
1 String sql = "SELECT oyuncu, puan FROM skorlar "  
2       + "WHERE puan > 70 "  
3       + "ORDER BY oyuncu DESC LIMIT 2";  
4 try (Statement st = conn.createStatement();  
5     ▶   ResultSet rs = st.executeQuery(sql)) {  
6     while (rs.next()) {  
7         System.out.println(rs.getString("oyuncu") + " "  
8                               + rs.getInt("puan"));  
9     }  
10 }
```

FROM

WHERE

ORDER BY

LIMIT

oyuncu	puan
ada	72
can	95
ece	58
efe	95
naz	81

ece 58 elendi

WHERE puan > 70 : koşulu sağlamayan
ece elenir; 4 satır kalır.

Adım 3 / 10

DEĞİŞKENLER

aşama = WHERE

ÇIKTI



Bir **SELECT** Sorgusu Nasıl Çalışır?

```
1 String sql = "SELECT oyuncu, puan FROM skorlar "  
2     + "WHERE puan > 70 "  
3     + "ORDER BY oyuncu DESC LIMIT 2";  
4 try (Statement st = conn.createStatement();  
5     ▶ ResultSet rs = st.executeQuery(sql)) {  
6     while (rs.next()) {  
7         System.out.println(rs.getString("oyuncu") + " "  
8             + rs.getInt("puan"));  
9     }  
10 }
```

FROM**WHERE****ORDER BY****LIMIT**

oyuncu	puan
naz	81
efe	95
can	95
ada	72

ad, Z → A

ORDER BY oyuncu DESC : adlar ters alfabetik sıraya dizilir (puana göre değil).

Adım 4 / 10

DEĞİŞKENLER

aşama = ORDER BY

ÇIKTI



Bir **SELECT** Sorgusu Nasıl Çalışır?

```
1 String sql = "SELECT oyuncu, puan FROM skorlar "  
2           + "WHERE puan > 70 "  
3           + "ORDER BY oyuncu DESC LIMIT 2";  
4 try (Statement st = conn.createStatement();  
5     ▶ ResultSet rs = st.executeQuery(sql)) {  
6     while (rs.next()) {  
7         System.out.println(rs.getString("oyuncu") + " "  
8                             + rs.getInt("puan"));  
9     }  
10 }
```

FROM**WHERE****ORDER BY****LIMIT**

oyuncu	puan
naz	81
efe	95
can	95
ada	72

ilk 2 satır

LIMIT 2 : sıralanmış listenin ilk iki satırı sonuç kümesidir.

Adım 5 / 10

DEĞİŞKENLER

aşama = **LIMIT**

ÇIKTI



Bir **SELECT** Sorgusu Nasıl Çalışır?

```
1 String sql = "SELECT oyuncu, puan FROM skorlar "  
2     + "WHERE puan > 70 "  
3     + "ORDER BY oyuncu DESC LIMIT 2";  
4 try (Statement st = conn.createStatement();  
5     ResultSet rs = st.executeQuery(sql)) {  
6     ▶ while (rs.next()) {  
7         System.out.println(rs.getString("oyuncu") + " "  
8             + rs.getInt("puan"));  
9     }  
10 }
```

	FROM	WHERE	ORDER BY	LIMIT
imleç				
	oyuncu	puan		
	naz	81		
	efe	95		

sonuç kümesi

İmleç başlangıçta ilk satırın **önündedir**;
`next()` onu 1. satıra taşır.

Adım 6 / 10

DEĞİŞKENLER

`rs.next() = true`

ÇIKTI

Bir **SELECT** Sorgusu Nasıl Çalışır?

```
1 String sql = "SELECT oyuncu, puan FROM skorlar "  
2     + "WHERE puan > 70 "  
3     + "ORDER BY oyuncu DESC LIMIT 2";  
4 try (Statement st = conn.createStatement();  
5     ResultSet rs = st.executeQuery(sql)) {  
6     while (rs.next()) {  
7         ▶ System.out.println(rs.getString("oyuncu") + " "  
8         ▶                               + rs.getInt("puan"));  
9     }  
10 }
```

FROM

WHERE

ORDER BY

LIMIT

imleç
→

oyuncu	puan
naz	81
efe	95

sonuç kümesi

Sütunlar adlarıyla okunur ve yazdırılır.

Adım 7 / 10

DEĞİŞKENLER

oyuncu = naz

puan = 81

ÇIKTI

naz 81

Bir **SELECT** Sorgusu Nasıl Çalışır?

```
1 String sql = "SELECT oyuncu, puan FROM skorlar "  
2     + "WHERE puan > 70 "  
3     + "ORDER BY oyuncu DESC LIMIT 2";  
4 try (Statement st = conn.createStatement();  
5     ResultSet rs = st.executeQuery(sql)) {  
6     ▶ while (rs.next()) {  
7         System.out.println(rs.getString("oyuncu") + " "  
8             + rs.getInt("puan"));  
9     }  
10 }
```

FROM

WHERE

ORDER BY

LIMIT

imleç
→

oyuncu	puan
naz	81
efer	95

sonuç kümesi

imleç 2. satıra geçer.

Adım 8 / 10

DEĞİŞKENLER

`rs.next() = true`

ÇIKTI

naz 81

Bir **SELECT** Sorgusu Nasıl Çalışır?

```
1 String sql = "SELECT oyuncu, puan FROM skorlar "  
2     + "WHERE puan > 70 "  
3     + "ORDER BY oyuncu DESC LIMIT 2";  
4 try (Statement st = conn.createStatement();  
5     ResultSet rs = st.executeQuery(sql)) {  
6     while (rs.next()) {  
7         ▶ System.out.println(rs.getString("oyuncu") + " "  
8         ▶                               + rs.getInt("puan"));  
9     }  
10 }
```

FROM

WHERE

ORDER BY

LIMIT

imleç

oyuncu	puan
naz	81
efer	95

sonuç kümesi

efer 95 yazılır.

Adım 9 / 10

DEĞİŞKENLER

oyuncu = efer

puan = 95

ÇIKTI

naz 81

efer 95

Bir **SELECT** Sorgusu Nasıl Çalışır?

```
1 String sql = "SELECT oyuncu, puan FROM skorlar "  
2     + "WHERE puan > 70 "  
3     + "ORDER BY oyuncu DESC LIMIT 2";  
4 try (Statement st = conn.createStatement();  
5     ResultSet rs = st.executeQuery(sql)) {  
▶   while (rs.next()) {  
7       System.out.println(rs.getString("oyuncu") + " "  
8           + rs.getInt("puan"));  
9   }  
▶ }
```

FROM

WHERE

ORDER BY

LIMIT

oyuncu	puan
naz	81
efe	95

sonuç kümesi

imleç
→

Başka satır yok: `next()` `false`, döngü biter. `try` bitince `rs` ve `st` kapanır.

Adım 10 / 10

DEĞİŞKENLER

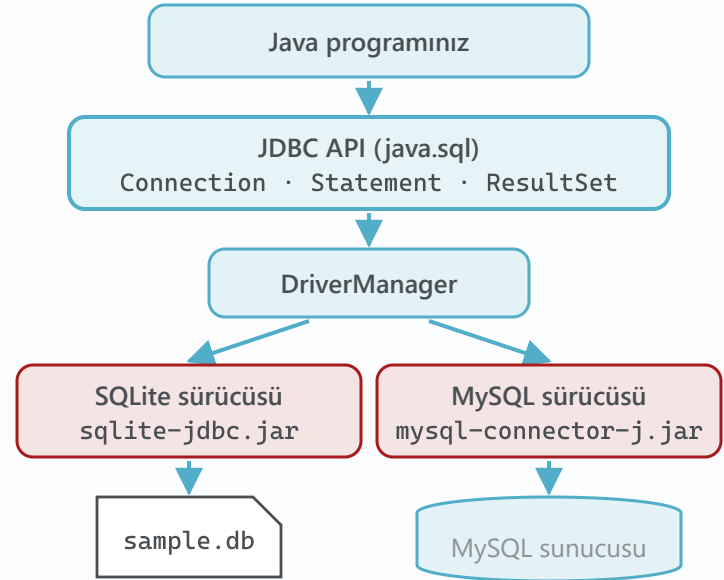
`rs.next() = false`

ÇIKTI

```
naz 81  
efe 95
```

JDBC (Java Database Connectivity)

- **JDBC** (Java Database Connectivity): Java'nın veritabanlarına bağlanmak için sunduğu **standart API**'dir; `java.sql` paketindedir ve JDK ile gelir.
- Program yalnızca JDBC **arayüzlerini** kullanır; her veri tabanı üreticisi bu arayüzleri gerçekleştiren bir **sürücü** (driver) sağlar.
- Veri tabanını değiştirmek için çoğu zaman yalnızca **sürücü jar'ı** ve **bağlantı adresi** değişir.
- Çok biçimlilik iş başında: `Connection` bir arayüzdür; SQLite'ta gerçek nesnenin sınıfı `org.sqlite.jdbc4.JDBC4Connection` 'dır.



JDBC'nin Temel Türleri

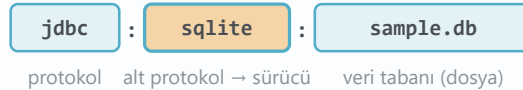
Tür	Görevi	Önemli metotlar
DriverManager	Uygun sürücüyü bulur, bağlantı açar	getConnection(url) , getConnection(url, kullanıcı, şifre)
Connection	Veri tabanıyla açık bir oturum	createStatement() , preparedStatement(sql) , commit() , rollback() , close()
Statement	Sabit bir SQL metnini çalıştırır	execute , executeUpdate , executeQuery
PreparedStatement	? yer tutuculu, önceden hazırlanan SQL	setInt , setString , executeUpdate() , executeQuery()
ResultSet	Sorgu sonucundaki satırlar ve bir imleç	next() , getInt("id") , getString("name")
SQLException	Veri tabanı hatası; kontrol edilen istisna	getMessage() , getErrorCode()

Hepsi `java.sql` paketindedir: dosyanın başına `import java.sql.*;` yazılır.

`Connection` , `Statement` ve `ResultSet` kapatılması gereken kaynaklardır (`AutoCloseable`).

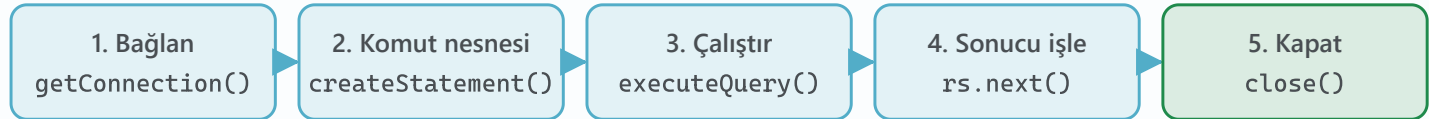
Bağlantı Adresi (JDBC URL)

Bağlantı adresi (URL)	Anlamı
<code>jdbc:sqlite:sample.db</code>	Çalışma dizinindeki <code>sample.db</code> ; yoksa boş olarak oluşturulur
<code>jdbc:sqlite:C:/veriler/okul.db</code>	Tam yol ile dosya
<code>jdbc:sqlite::memory:</code>	Bellekte geçici veri tabanı; bağlantı kapanınca silinir (denemeler için)
<code>jdbc:mysql://localhost:3306/testdb</code>	MySQL sunucusu: makine, port, veri tabanı adı; ayrıca kullanıcı adı ve şifre



- Biçim: `jdbc:<alt protokol>:<veritabanı_dosyanız veya adresi>` . `DriverManager` , alt protokole bakarak sürücüyü seçer.
- JDBC 4.0 (Java 6) ve sonrasında sürücü, classpath'teki jar'dan **otomatik** yüklenir. Eski kodlardaki `Class.forName("org.sqlite.JDBC")` satırı artık gerekmez.
- Sürücü classpath'te yoksa: `No suitable driver found for jdbc:sqlite:sample.db` .

Bir JDBC Programının Adımları



```
try (Connection conn = DriverManager.getConnection(url); // 1
    Statement stmt = conn.createStatement();           // 2
    ResultSet rs = stmt.executeQuery(sql)) {           // 3
    while (rs.next()) {                                // 4
        System.out.println(rs.getInt(1));
    }
} // 5: rs, stmt, conn ters sırada otomatik kapanır
```

- Kaynaklar **try-with-resources** parantezinde açılır; blok bitince (hata olsa bile) **ters sırada** kapanır: önce `rs`, sonra `stmt`, en son `conn`.
- Kapatılmayan bağlantılar dosyayı kilitli bırakabilir ve bellek sızıntısına yol açar.
- `getInt(1)` : sütun numaraları **1'den** başlar.

TEMEL İŞLEMLER

SQLite'a Bağlanma

```
// Veritabanı dosyasının yolu (bağlantı adresi)
String url = "jdbc:sqlite:sample.db";
try (Connection conn = DriverManager.getConnection(url)) {
    // Bağlantı başarılıysa mesaj yazdırılır.
    if (conn != null) {
        System.out.println("Bağlantı başarılı!");
    }
} catch (SQLException e) {
    // Bağlantı hatası olursa hata mesajı yazdırılır.
    System.out.println(e.getMessage());
}
```

ÇIKTI

Bağlantı başarılı!

- `getConnection` dosya yoksa **oluşturur** (boş veri tabanı).
- Bağlantı kurulamazsa `null` dönmez, `SQLException` fırlatır; `if (conn != null)` yalnızca ek bir güvencedir.
- `SQLException` kontrol edilen istisnadır: yakalanmalı ya da `throws` ile bildirilmelidir.

Tablo Oluşturma

```
String url = "jdbc:sqlite:sample.db";
String sql = "CREATE TABLE IF NOT EXISTS users "
    + "(id INTEGER PRIMARY KEY, name TEXT, email TEXT)";
try (Connection conn = DriverManager.getConnection(url);
    Statement stmt = conn.createStatement()) {
    // Tablo oluşturuluyor.
    stmt.execute(sql);
    System.out.println("Tablo oluşturuldu!");
} catch (SQLException e) {
    System.out.println(e.getMessage());
}
```

ÇIKTI

Tablo oluşturuldu!

- `execute(sql)` her tür SQL'i çalıştırır; sonuç kümesi üretirse `true`, `CREATE` için `false` döner.
- `IF NOT EXISTS` sayesinde program ikinci kez çalışınca hata olmaz. Olmasaydı: `table users already exists`.
- Birden çok kaynak tek `try (...; ...)` içinde `;` ile ayrılır.

Veri Ekleme

```
String url = "jdbc:sqlite:sample.db";
String sql = "INSERT INTO users(name, email) "
            + "VALUES('Ahmet Yılmaz', 'ahmet@example.com')";
try (Connection conn = DriverManager.getConnection(url);
     Statement stmt = conn.createStatement()) {
    // Veritabanına veri ekleniyor.
    int sayi = stmt.executeUpdate(sql);
    System.out.println("Veri eklendi! (" + sayi + " satır)");
} catch (SQLException e) {
    System.out.println(e.getMessage());
}
```

ÇIKTI

Veri eklendi! (1 satır)

- `executeUpdate` **etkilenen satır sayısını** (`int`) döndürür.
- `id` verilmedi; SQLite `1` atadı.
- Program her çalıştığında **yeni bir satır** eklenir (id 2, 3, ...).
- SQL içindeki metin değerler tek tırnak içindedir.

Veri Okuma

```
String url = "jdbc:sqlite:sample.db";
String sql = "SELECT * FROM users";
try (Connection conn = DriverManager.getConnection(url);
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery(sql)) {
    // Satırlar tek tek okunup yazdırılıyor.
    while (rs.next()) {
        System.out.println("ID: " + rs.getInt("id")
            + ", Ad: " + rs.getString("name")
            + ", E-posta: " + rs.getString("email"));
    }
} catch (SQLException e) {
    System.out.println(e.getMessage());
}
```

ÇIKTI

ID: 1, Ad: Ahmet Yılmaz, E-posta:
ahmet@example.com

- `executeQuery` bir `ResultSet` döndürür.
- `rs.next()` imleci bir sonraki satıra taşır; satır varsa `true`, yoksa `false` döner.
- `getInt("id")`,
`getString("name")` : o anki satırın sütunlarını adıyla okur.

ResultSet ile Okuma: Adım Adım

```
1 String url = "jdbc:sqlite:okuma.db";
2 String sql = "SELECT * FROM users";
▶ try (Connection conn = DriverManager.getConnection(url);
▶     Statement stmt = conn.createStatement();
▶     ResultSet rs = stmt.executeQuery(sql)) {
6     while (rs.next()) {
7         int id = rs.getInt("id");
8         String ad = rs.getString("name");
9         System.out.println(id + " - " + ad);
10    }
11    System.out.println("Okuma bitti.");
12 } catch (SQLException e) {
13     System.out.println(e.getMessage());
14 }
```

ResultSet rs



id	name
1	Ahmet Yılmaz
2	Ayşe Demir

imleç: ilk satırdan önce

Bağlantı açılır, sorgu çalışır. `rs` iki satırlık sonucu tutar; imleç **ilk satırdan önce** bekler.

Adım 1 / 10

ÇIKTI



DEĞİŞKENLER

—

ResultSet ile Okuma: Adım Adım

```
1 String url = "jdbc:sqlite:okuma.db";
2 String sql = "SELECT * FROM users";
3 try (Connection conn = DriverManager.getConnection(url);
4     Statement stmt = conn.createStatement();
5     ResultSet rs = stmt.executeQuery(sql)) {
6     ▶ while (rs.next()) {
7         int id = rs.getInt("id");
8         String ad = rs.getString("name");
9         System.out.println(id + " - " + ad);
10    }
11    System.out.println("Okuma bitti.");
12 } catch (SQLException e) {
13    System.out.println(e.getMessage());
14 }
```

ResultSet rs



id	name
1	Ahmet Yılmaz
2	Ayşe Demir

imleç: 1. satır

`rs.next()` imleci 1. satıra taşır ve
`true` döndürür: döngüye girilir.

Adım 2 / 10

ÇIKTI



DEĞİŞKENLER

`rs.next() = true`

ResultSet ile Okuma: Adım Adım

```
1 String url = "jdbc:sqlite:okuma.db";
2 String sql = "SELECT * FROM users";
3 try (Connection conn = DriverManager.getConnection(url);
4     Statement stmt = conn.createStatement();
5     ResultSet rs = stmt.executeQuery(sql)) {
6     while (rs.next()) {
7         ▶ int id = rs.getInt("id");
8         ▶ String ad = rs.getString("name");
9         System.out.println(id + " - " + ad);
10    }
11    System.out.println("Okuma bitti.");
12 } catch (SQLException e) {
13    System.out.println(e.getMessage());
14 }
```

ResultSet rs



id	name
1	Ahmet Yılmaz
2	Ayşe Demir

imleç: 1. satır

Sütunlar **o anki satırdan** okunur.

Adım 3 / 10

ÇIKTI



DEĞİŞKENLER

rs.next() = true

id = 1

ad = "Ahmet Yılmaz"

ResultSet ile Okuma: Adım Adım

```
1 String url = "jdbc:sqlite:okuma.db";
2 String sql = "SELECT * FROM users";
3 try (Connection conn = DriverManager.getConnection(url);
4     Statement stmt = conn.createStatement();
5     ResultSet rs = stmt.executeQuery(sql)) {
6     while (rs.next()) {
7         int id = rs.getInt("id");
8         String ad = rs.getString("name");
9         ▶ System.out.println(id + " - " + ad);
10    }
11    System.out.println("Okuma bitti.");
12 } catch (SQLException e) {
13     System.out.println(e.getMessage());
14 }
```

ResultSet rs



id	name
1	Ahmet Yılmaz
2	Ayşe Demir

imleç: 1. satır

Satır yazdırılır.

Adım 4 / 10

ÇIKTI

```
1 - Ahmet Yılmaz
```

DEĞİŞKENLER

rs.next() = true

id = 1

ad = "Ahmet Yılmaz"

ResultSet ile Okuma: Adım Adım

```
1 String url = "jdbc:sqlite:okuma.db";
2 String sql = "SELECT * FROM users";
3 try (Connection conn = DriverManager.getConnection(url);
4     Statement stmt = conn.createStatement();
5     ResultSet rs = stmt.executeQuery(sql)) {
6     ▶ while (rs.next()) {
7         int id = rs.getInt("id");
8         String ad = rs.getString("name");
9         System.out.println(id + " - " + ad);
10    }
11    System.out.println("Okuma bitti.");
12 } catch (SQLException e) {
13     System.out.println(e.getMessage());
14 }
```

ResultSet rs

id	name
1	Ahmet Yılmaz
2	Ayşe Demir

rs →

imleç: 2. satır

İmleç 2. satıra geçer; yine `true`.

Adım 5 / 10

ÇIKTI

1 - Ahmet Yılmaz

DEĞİŞKENLER

rs.next() = `true`

id = `1`

ad = `"Ahmet Yılmaz"`

ResultSet ile Okuma: Adım Adım

```
1 String url = "jdbc:sqlite:okuma.db";
2 String sql = "SELECT * FROM users";
3 try (Connection conn = DriverManager.getConnection(url);
4     Statement stmt = conn.createStatement();
5     ResultSet rs = stmt.executeQuery(sql)) {
6     while (rs.next()) {
7         ▶ int id = rs.getInt("id");
8         ▶ String ad = rs.getString("name");
9         System.out.println(id + " - " + ad);
10    }
11    System.out.println("Okuma bitti.");
12 } catch (SQLException e) {
13    System.out.println(e.getMessage());
14 }
```

ResultSet rs



id	name
1	Ahmet Yılmaz
2	Ayşe Demir

imleç: 2. satır

Bu kez 2. satırın değerleri okunur.

Adım 6 / 10

ÇIKTI

1 - Ahmet Yılmaz

DEĞİŞKENLER

rs.next() = true

id = 2

ad = "Ayşe Demir"

ResultSet ile Okuma: Adım Adım

```
1 String url = "jdbc:sqlite:okuma.db";
2 String sql = "SELECT * FROM users";
3 try (Connection conn = DriverManager.getConnection(url);
4     Statement stmt = conn.createStatement();
5     ResultSet rs = stmt.executeQuery(sql)) {
6     while (rs.next()) {
7         int id = rs.getInt("id");
8         String ad = rs.getString("name");
9         ▶ System.out.println(id + " - " + ad);
10    }
11    System.out.println("Okuma bitti.");
12 } catch (SQLException e) {
13     System.out.println(e.getMessage());
14 }
```

ResultSet rs

id	name
1	Ahmet Yılmaz
2	Ayşe Demir



imleç: 2. satır

İkinci satır yazdırılır.

Adım 7 / 10

ÇIKTI

```
1 - Ahmet Yılmaz
2 - Ayşe Demir
```

DEĞİŞKENLER

rs.next() = true

id = 2

ad = "Ayşe Demir"

ResultSet ile Okuma: Adım Adım

```
1 String url = "jdbc:sqlite:okuma.db";
2 String sql = "SELECT * FROM users";
3 try (Connection conn = DriverManager.getConnection(url);
4     Statement stmt = conn.createStatement();
5     ResultSet rs = stmt.executeQuery(sql)) {
6     ▶ while (rs.next()) {
7         int id = rs.getInt("id");
8         String ad = rs.getString("name");
9         System.out.println(id + " - " + ad);
10    }
11    System.out.println("Okuma bitti.");
12 } catch (SQLException e) {
13     System.out.println(e.getMessage());
14 }
```

ResultSet rs

id	name
1	Ahmet Yılmaz
2	Ayşe Demir



imleç: son satırdan sonra

Başka satır yok: `rs.next()` `false`
döner, döngü biter.

Adım 8 / 10

ÇIKTI

```
1 - Ahmet Yılmaz
2 - Ayşe Demir
```

DEĞİŞKENLER

`rs.next()` = `false`

ResultSet ile Okuma: Adım Adım

```
1 String url = "jdbc:sqlite:okuma.db";
2 String sql = "SELECT * FROM users";
3 try (Connection conn = DriverManager.getConnection(url);
4     Statement stmt = conn.createStatement();
5     ResultSet rs = stmt.executeQuery(sql)) {
6     while (rs.next()) {
7         int id = rs.getInt("id");
8         String ad = rs.getString("name");
9         System.out.println(id + " - " + ad);
10    }
11    ▶ System.out.println("Okuma bitti.");
12 } catch (SQLException e) {
13     System.out.println(e.getMessage());
14 }
```

ResultSet rs

id	name
1	Ahmet Yılmaz
2	Ayşe Demir



imleç: son satırdan sonra

Döngüden sonraki satır çalışır.

Adım 9 / 10

ÇIKTI

```
1 - Ahmet Yılmaz
2 - Ayşe Demir
Okuma bitti.
```

DEĞİŞKENLER

rs.next() = **false**

ResultSet ile Okuma: Adım Adım

```
1 String url = "jdbc:sqlite:okuma.db";
2 String sql = "SELECT * FROM users";
3 try (Connection conn = DriverManager.getConnection(url);
4     Statement stmt = conn.createStatement();
5     ResultSet rs = stmt.executeQuery(sql)) {
6     while (rs.next()) {
7         int id = rs.getInt("id");
8         String ad = rs.getString("name");
9         System.out.println(id + " - " + ad);
10    }
11    System.out.println("Okuma bitti.");
12 } catch (SQLException e) {
13     System.out.println(e.getMessage());
14 }
```

ResultSet rs

id	name
1	Ahmet Yılmaz
2	Ayşe Demir



imleç: son satırdan sonra

try bloğu biter: `rs`, `stmt`, `conn`
bu sırayla **otomatik kapanır**.

Adım 10 / 10

ÇIKTI

```
1 - Ahmet Yılmaz
2 - Ayşe Demir
Okuma bitti.
```

DEĞİŞKENLER

—

Veri Güncelleme

```
String url = "jdbc:sqlite:sample.db";
String sql = "UPDATE users SET name = 'Mehmet Akdeniz' WHERE id = 1";
try (Connection conn = DriverManager.getConnection(url);
    Statement stmt = conn.createStatement()) {
    // Veritabanındaki veri güncelleniyor.
    int sayi = stmt.executeUpdate(sql);
    System.out.println("Veri güncellendi! (" + sayi + " satır)");
} catch (SQLException e) {
    System.out.println(e.getMessage());
}
```

ÇIKTI

Veri güncellendi! (1 satır)

- Öncesinde okuma: ID: 1, Ad: Ahmet Yılmaz, ... ;
sonrasında: ID: 1, Ad: Mehmet Akdeniz, ...
- SET yalnızca yazılan sütunu değiştirir; email aynı kalır.
- WHERE id = 1 güncellenecek satırı seçer. **WHERE unutulursa tüm satırlar** güncellenir.

Veri Silme

```
String url = "jdbc:sqlite:sample.db";
String sql = "DELETE FROM users WHERE id = 1";
try (Connection conn = DriverManager.getConnection(url);
     Statement stmt = conn.createStatement()) {
    // Veritabanındaki veri siliniyor.
    int sayi = stmt.executeUpdate(sql);
    System.out.println("Veri silindi! (" + sayi + " satır)");
} catch (SQLException e) {
    System.out.println(e.getMessage());
}
```

ÇIKTI (PROGRAM İKİ KEZ ÇALIŞTIRILINCA)

```
Veri silindi! (1 satır)
Veri silindi! (0 satır)
```

- İkinci çalıştırmada silinecek satır kalmamıştır: hata oluşmaz, `executeUpdate` `0` döner.
- Bu yüzden mesajı dönen sayıya göre vermek daha doğrudur (Alıştırma 1).
- WHERE'siz `DELETE FROM users` **tüm** satırları siler; tablonun kendisi kalır.

TEMEL İŞLEMLER

execute, executeUpdate, executeQuery

Metot	Döndürür	Kullanım
<code>execute</code>	<code>boolean</code>	<code>CREATE</code> <code>TABLE</code> , <code>DROP</code>
<code>executeUpdate</code>	<code>int</code> satır sayısı	<code>INSERT</code> , <code>UPDATE</code> , <code>DELETE</code>
<code>executeQuery</code>	<code>ResultSet</code>	<code>SELECT</code>

Yanlış metot hataya yol açar: `executeQuery` ile `INSERT` çalıştırmak → `query does not return ResultSet`.

PROGRAMLAR SIRAYLA ÇALIŞTIRILINCA

```
> java Baglan
Bağlantı başarılı!
> java TabloOlustur
Tablo oluşturuldu!
> java VeriEkle
Veri eklendi! (1 satır)
> java VeriOku
ID: 1, Ad: Ahmet Yılmaz, E-posta:
ahmet@example.com
> java VeriGuncelle
Veri güncellendi! (1 satır)
> java VeriOku
ID: 1, Ad: Mehmet Akdeniz, E-posta:
ahmet@example.com
> java VeriSil
Veri silindi! (1 satır)
> java VeriOku
> java VeriSil
Veri silindi! (0 satır)
```

executeUpdate Ne Döndürür?

```
▶ String[] adlar = {"zımba", "makas", "bant"};
▶ int[] stok = {40, 0, 15};
▶ PreparedStatement ek = conn.prepareStatement(
    ▶ "INSERT INTO malzeme(ad, stok) VALUES(?, ?)");
5 for (int i = 0; i < adlar.length; i++) {
6     ek.setString(1, adlar[i]);
7     ek.setInt(2, stok[i]);
8     ek.executeUpdate();
9 }
10 int g = stmt.executeUpdate(
11     "UPDATE malzeme SET stok = stok + 10 WHERE stok <
12     20");
12 int s = stmt.executeUpdate("DELETE FROM malzeme WHERE
13     stok > 100");
13 System.out.println(g + " güncellendi, " + s + "
    silindi");
```

malzeme

id	ad	stok
----	----	------

tablo boş

Ekleme komutu **bir kez** hazırlanır; döngüde yalnızca değerler değişecek.

Adım 1 / 8

DEĞİŞKENLER

adlar = [zımba, makas, bant]

stok = [40, 0, 15]

ÇIKTI



executeUpdate Ne Döndürür?

```
1 String[] adlar = {"zımba", "makas", "bant"};
2 int[] stok = {40, 0, 15};
3 PreparedStatement ek = conn.prepareStatement(
4     "INSERT INTO malzeme(ad, stok) VALUES(?, ?)");
5 for (int i = 0; i < adlar.length; i++) {
6     ▶ ek.setString(1, adlar[i]);
7     ▶ ek.setInt(2, stok[i]);
8     ek.executeUpdate();
9 }
10 int g = stmt.executeUpdate(
11     "UPDATE malzeme SET stok = stok + 10 WHERE stok <
12     20");
13 int s = stmt.executeUpdate("DELETE FROM malzeme WHERE
14     stok > 100");
15 System.out.println(g + " güncellendi, " + s + "
16     silindi");
```

malzeme

id	ad	stok
----	----	------

yer tutucular

1. ? = "zımba"

2. ? = 40

İlk turda yer tutucular doldurulur.

Adım 2 / 8

DEĞİŞKENLER

i = 0

ÇIKTI

executeUpdate Ne Döndürür?

```
1 String[] adlar = {"zımba", "makas", "bant"};
2 int[] stok = {40, 0, 15};
3 PreparedStatement ek = conn.prepareStatement(
4     "INSERT INTO malzeme(ad, stok) VALUES(?, ?)");
5 for (int i = 0; i < adlar.length; i++) {
6     ek.setString(1, adlar[i]);
7     ek.setInt(2, stok[i]);
8     ▶ ek.executeUpdate();
9 }
10 int g = stmt.executeUpdate(
11     "UPDATE malzeme SET stok = stok + 10 WHERE stok <
12     20");
13 int s = stmt.executeUpdate("DELETE FROM malzeme WHERE
14     stok > 100");
15 System.out.println(g + " güncellendi, " + s + "
16     silindi");
```

malzeme

id	ad	stok
1	zımba	40

1 satır eklendi

executeUpdate 1 döndürür; SQLite id = 1 atar.

Adım 3 / 8

DEĞİŞKENLER

i = 0

dönen = 1

ÇIKTI



executeUpdate Ne Döndürür?

```
1 String[] adlar = {"zımba", "makas", "bant"};
2 int[] stok = {40, 0, 15};
3 PreparedStatement ek = conn.prepareStatement(
4     "INSERT INTO malzeme(ad, stok) VALUES(?, ?)");
5 ▶ for (int i = 0; i < adlar.length; i++) {
6     ▶ ek.setString(1, adlar[i]);
7     ▶ ek.setInt(2, stok[i]);
8     ▶ ek.executeUpdate();
9 }
10 int g = stmt.executeUpdate(
11     "UPDATE malzeme SET stok = stok + 10 WHERE stok <
12     20");
13 int s = stmt.executeUpdate("DELETE FROM malzeme WHERE
14     stok > 100");
15 System.out.println(g + " güncellendi, " + s + "
16     silindi");
```

malzeme

id	ad	stok
1	zımba	40
2	makas	0

yer tutucular

1. ? = "makas"

2. ? = 0

1 satır eklendi

Aynı hazır komut yeni değerlerle yeniden çalışır.

Adım 4 / 8

DEĞİŞKENLER

i = 1

dönen = 1

ÇIKTI



executeUpdate Ne Döndürür?

```
1 String[] adlar = {"zımba", "makas", "bant"};
2 int[] stok = {40, 0, 15};
3 PreparedStatement ek = conn.prepareStatement(
4     "INSERT INTO malzeme(ad, stok) VALUES(?, ?)");
5 ▶ for (int i = 0; i < adlar.length; i++) {
6     ▶ ek.setString(1, adlar[i]);
7     ▶ ek.setInt(2, stok[i]);
8     ▶ ek.executeUpdate();
9 }
10 int g = stmt.executeUpdate(
11     "UPDATE malzeme SET stok = stok + 10 WHERE stok <
12     20");
13 int s = stmt.executeUpdate("DELETE FROM malzeme WHERE
14     stok > 100");
15 System.out.println(g + " güncellendi, " + s + "
16     silindi");
```

malzeme

id	ad	stok
1	zımba	40
2	makas	0
3	bant	15

yer tutucular

1. ? = "bant"

2. ? = 15

1 satır eklendi

Üçüncü ürün eklenir; **i** = 3 olunca
döngü biter.

Adım 5 / 8

DEĞİŞKENLER

i = 2**dönen = 1**

ÇIKTI



executeUpdate Ne Döndürür?

```
1 String[] adlar = {"zımba", "makas", "bant"};
2 int[] stok = {40, 0, 15};
3 PreparedStatement ek = conn.prepareStatement(
4     "INSERT INTO malzeme(ad, stok) VALUES(?, ?)");
5 for (int i = 0; i < adlar.length; i++) {
6     ek.setString(1, adlar[i]);
7     ek.setInt(2, stok[i]);
8     ek.executeUpdate();
9 }
10 ▶ int g = stmt.executeUpdate(
11     "UPDATE malzeme SET stok = stok + 10 WHERE stok <
12     20");
13 int s = stmt.executeUpdate("DELETE FROM malzeme WHERE
14     stok > 100");
15 System.out.println(g + " güncellendi, " + s + "
16     silindi");
```

malzeme

id	ad	stok
1	zımba	40
2	makas	10
3	bant	25

WHERE stok < 20: 2 satır

WHERE stok < 20 makas (0) ve bandı (15) seçer; ikisine 10 eklenir. Dönen değer **etkilenen satır sayısı: 2.**

Adım 6 / 8

DEĞİŞKENLER

g = 2

ÇIKTI



executeUpdate Ne Döndürür?

```
1 String[] adlar = {"zımba", "makas", "bant"};
2 int[] stok = {40, 0, 15};
3 PreparedStatement ek = conn.prepareStatement(
4     "INSERT INTO malzeme(ad, stok) VALUES(?, ?)");
5 for (int i = 0; i < adlar.length; i++) {
6     ek.setString(1, adlar[i]);
7     ek.setInt(2, stok[i]);
8     ek.executeUpdate();
9 }
10 int g = stmt.executeUpdate(
11     "UPDATE malzeme SET stok = stok + 10 WHERE stok <
12     20");
13 ▶ int s = stmt.executeUpdate("DELETE FROM malzeme WHERE
14     stok > 100");
15 System.out.println(g + " güncellendi, " + s + "
16     silindi");
```

malzeme

id	ad	stok
1	zımba	40
2	makas	10
3	bant	25

eşleşen yok: 0

Stoku 100'den büyük ürün yok: hata oluşmaz, yalnızca 0 döner.

Adım 7 / 8

DEĞİŞKENLER

g = 2

s = 0

ÇIKTI

executeUpdate Ne Döndürür?

```
1 String[] adlar = {"zımba", "makas", "bant"};
2 int[] stok = {40, 0, 15};
3 PreparedStatement ek = conn.prepareStatement(
4     "INSERT INTO malzeme(ad, stok) VALUES(?, ?)");
5 for (int i = 0; i < adlar.length; i++) {
6     ek.setString(1, adlar[i]);
7     ek.setInt(2, stok[i]);
8     ek.executeUpdate();
9 }
10 int g = stmt.executeUpdate(
11     "UPDATE malzeme SET stok = stok + 10 WHERE stok <
12     20");
13 int s = stmt.executeUpdate("DELETE FROM malzeme WHERE
14     stok > 100");
15 System.out.println(g + " güncellendi, " + s + "
16     silindi");
```

malzeme

id	ad	stok
1	zımba	40
2	makas	10
3	bant	25

Program bu sayılara bakarak "kayıt bulunamadı" gibi doğru mesajlar verebilir.

Adım 8 / 8

DEĞİŞKENLER

g = 2

s = 0

ÇIKTI

2 güncellendi, 0 silindi

SQL Enjeksiyonu (SQL Injection)

```
String girdi = "x' OR '1'='1"; // kullanıcının yazdığı "ad"

// TEHLİKELİ: girdi SQL metnine yapıştırılıyor
String sql = "SELECT ad FROM kisiler WHERE ad = '"
            + girdi + "'";
System.out.println(sql);
ResultSet rs1 = stmt.executeQuery(sql);

// GÜVENLİ: girdi yalnızca değer olarak gönderiliyor
PreparedStatement ps = conn.prepareStatement(
    "SELECT ad FROM kisiler WHERE ad = ?");
ps.setString(1, girdi);
ResultSet rs2 = ps.executeQuery();
```

ÇIKTI (TABLODA 3 KİŞİ VAR; SATIRLAR SAYILDI)

```
SELECT ad FROM kisiler WHERE ad
= 'x' OR '1'='1'
Birleştirme: 3 satır
PreparedStatement: 0 satır
```

- Kullanıcının yazdığı tırnak, SQL metnini **değiştirdi**: `'1'='1'` her zaman doğru, tüm satırlar döndü.
- Giriş ekranında bu, **şifresiz oturum açmak** demektir.
- `?` ile gönderilen girdi yalnızca bir **değerdir**; SQL'in parçası olamaz.

SQL Enjeksiyonu Adım Adım

```
► String girdi = "x' OR '1'='1";
2 String sql = "SELECT ad FROM kisiler WHERE ad = '"
3       + girdi + "'";
4 int n = 0;
5 try (ResultSet rs = stmt.executeQuery(sql)) {
6     while (rs.next()) n++;
7 }
8 System.out.println("Birleştirme: " + n + " satır");
9 PreparedStatement ps = conn.prepareStatement(
10     "SELECT ad FROM kisiler WHERE ad = ?");
11 ps.setString(1, girdi);
12 n = 0;
13 try (ResultSet rs = ps.executeQuery()) { while
    (rs.next()) n++; }
14 System.out.println("PreparedStatement: " + n + "
    satır");
```

veri tabanına giden SQL:

(henüz yok)

id	ad
1	Ali
2	Ayşe
3	Can

Kullanıcı ad yerine tırnak ve **OR** içeren bir metin yazdı. Tabloda 3 kişi var.

Adım 1 / 10

DEĞİŞKENLER

girdi = x' OR '1'='1

ÇIKTI



SQL Enjeksiyonu Adım Adım

```
1 String girdi = "x' OR '1'='1";
▶ String sql = "SELECT ad FROM kisiler WHERE ad = '"
▶       + girdi + "'";
4 int n = 0;
5 try (ResultSet rs = stmt.executeQuery(sql)) {
6     while (rs.next()) n++;
7 }
8 System.out.println("Birleştirme: " + n + " satır");
9 PreparedStatement ps = conn.prepareStatement(
10     "SELECT ad FROM kisiler WHERE ad = ?");
11 ps.setString(1, girdi);
12 n = 0;
13 try (ResultSet rs = ps.executeQuery()) { while
    (rs.next()) n++; }
14 System.out.println("PreparedStatement: " + n + "
    satır");
```

veri tabanına giden SQL:

```
SELECT ad FROM kisiler
WHERE ad = 'x' OR '1'='1'
```

id	ad
1	Ali
2	Ayşe
3	Can

Birleştirme sonucu: kullanıcının tırnağı ad değerini **kapattı**, ardından gelen **OR '1'='1'** artık SQL'in parçası.

Adım 2 / 10

DEĞİŞKENLER

```
sql SELECT ... WHERE ad = 'x' OR
= '1'='1'
```

ÇIKTI



SQL Enjeksiyonu Adım Adım

```
1 String girdi = "x' OR '1'='1";
2 String sql = "SELECT ad FROM kisiler WHERE ad = '"
3           + girdi + "'";
4 int n = 0;
5 ▶ try (ResultSet rs = stmt.executeQuery(sql)) {
6     while (rs.next()) n++;
7 }
8 System.out.println("Birleştirme: " + n + " satır");
9 PreparedStatement ps = conn.prepareStatement(
10     "SELECT ad FROM kisiler WHERE ad = ?");
11 ps.setString(1, girdi);
12 n = 0;
13 try (ResultSet rs = ps.executeQuery()) { while
    (rs.next()) n++; }
14 System.out.println("PreparedStatement: " + n + "
    satır");
```

veri tabanına giden SQL:

```
SELECT ad FROM kisiler
WHERE ad = 'x' OR '1'='1'
```

id	ad
1	Ali
2	Ayşe
3	Can

koşul

'Ali'='x' → yanlış OR '1'='1' → DOĞRU

Veri tabanı koşulu her satırda değerlendirir:
Ali için 'un sağ tarafı her zaman doğru
→ satır seçilir.

Adım 3 / 10

DEĞİŞKENLER

rs = ResultSet

ÇIKTI



SQL Enjeksiyonu Adım Adım

```
1 String girdi = "x' OR '1'='1";
2 String sql = "SELECT ad FROM kisiler WHERE ad = '"
3             + girdi + "'";
4 int n = 0;
5 ▶ try (ResultSet rs = stmt.executeQuery(sql)) {
6     while (rs.next()) n++;
7 }
8 System.out.println("Birleştirme: " + n + " satır");
9 PreparedStatement ps = conn.prepareStatement(
10     "SELECT ad FROM kisiler WHERE ad = ?");
11 ps.setString(1, girdi);
12 n = 0;
13 try (ResultSet rs = ps.executeQuery()) { while
    (rs.next()) n++; }
14 System.out.println("PreparedStatement: " + n + "
    satır");
```

veri tabanına giden SQL:

```
SELECT ad FROM kisiler
WHERE ad = 'x' OR '1'='1'
```

id	ad
1	Ali
2	Ayşe
3	Can

koşul

'Ali'='x' → yanlış OR '1'='1' → DOĞRU

'Ayşe'='x' → yanlış OR doğru → DOĞRU

'Can'='x' → yanlış OR doğru → DOĞRU

Ayşe ve Can da seçilir: koşul hiçbir satırı elemez.

Adım 4 / 10

DEĞİŞKENLER

rs = ResultSet

ÇIKTI



SQL Enjeksiyonu Adım Adım

```
1 String girdi = "x' OR '1'='1";
2 String sql = "SELECT ad FROM kisiler WHERE ad = '"
3           + girdi + "'";
4 int n = 0;
5 try (ResultSet rs = stmt.executeQuery(sql)) {
6     ▶ while (rs.next()) n++;
7     ▶ }
8 System.out.println("Birleştirme: " + n + " satır");
9 PreparedStatement ps = conn.prepareStatement(
10     "SELECT ad FROM kisiler WHERE ad = ?");
11 ps.setString(1, girdi);
12 n = 0;
13 try (ResultSet rs = ps.executeQuery()) { while
14     (rs.next()) n++; }
15 System.out.println("PreparedStatement: " + n + "
16     satır");
```

veri tabanına giden SQL:

```
SELECT ad FROM kisiler
WHERE ad = 'x' OR '1'='1'
```

id	ad
1	Ali
2	Ayşe
3 satır döndü	

koşul

'Ali'='x' → yanlış OR '1'='1' → DOĞRU

'Ayşe'='x' → yanlış OR doğru → DOĞRU

'Can'='x' → yanlış OR doğru → DOĞRU

Döngü 3 kez döner. Bir giriş ekranında bu, şifresiz oturum açmak demektir.

Adım 5 / 10

DEĞİŞKENLER

n = 3

ÇIKTI



SQL Enjeksiyonu Adım Adım

```
1 String girdi = "x' OR '1'='1";
2 String sql = "SELECT ad FROM kisiler WHERE ad = '"
3             + girdi + "'";
4 int n = 0;
5 try (ResultSet rs = stmt.executeQuery(sql)) {
6     while (rs.next()) n++;
7 }
8
9 ▶ System.out.println("Birleştirme: " + n + " satır");
10 PreparedStatement ps = conn.prepareStatement(
11     "SELECT ad FROM kisiler WHERE ad = ?");
12 ps.setString(1, girdi);
13 n = 0;
14 try (ResultSet rs = ps.executeQuery()) { while
    (rs.next()) n++; }
15 System.out.println("PreparedStatement: " + n + "
    satır");
```

veri tabanına giden SQL:

```
SELECT ad FROM kisiler
WHERE ad = 'x' OR '1'='1'
```

id	ad
1	Ali
2	Ayşe
3 satır döndü	an

koşul

'Ali'='x' → yanlış OR '1'='1' → DOĞRU

'Ayşe'='x' → yanlış OR doğru → DOĞRU

'Can'='x' → yanlış OR doğru → DOĞRU

Sonuç yazılır.

Adım 6 / 10

DEĞİŞKENLER

n = 3

ÇIKTI

Birleştirme: 3 satır

SQL Enjeksiyonu Adım Adım

```
1 String girdi = "x' OR '1'='1";
2 String sql = "SELECT ad FROM kisiler WHERE ad = '"
3             + girdi + "'";
4 int n = 0;
5 try (ResultSet rs = stmt.executeQuery(sql)) {
6     while (rs.next()) n++;
7 }
8 System.out.println("Birleştirme: " + n + " satır");
▶ PreparedStatement ps = conn.prepareStatement(
▶     "SELECT ad FROM kisiler WHERE ad = ?");
11 ps.setString(1, girdi);
12 n = 0;
13 try (ResultSet rs = ps.executeQuery()) { while
    (rs.next()) n++; }
14 System.out.println("PreparedStatement: " + n + "
    satır");
```

veri tabanına giden SQL:

```
SELECT ad FROM kisiler
WHERE ad = ?
```

id	ad
1	Ali
2	Ayşe
3	Can

Bu kez SQL **önce** derlenir; **?** yalnızca bir değerin yeri. Komutun yapısı artık değişmez.

Adım 7 / 10

DEĞİŞKENLER

ps = PreparedStatement

ÇIKTI

Birleştirme: 3 satır

SQL Enjeksiyonu Adım Adım

```
1 String girdi = "x' OR '1'='1";
2 String sql = "SELECT ad FROM kisiler WHERE ad = '"
3           + girdi + "'";
4 int n = 0;
5 try (ResultSet rs = stmt.executeQuery(sql)) {
6     while (rs.next()) n++;
7 }
8 System.out.println("Birleştirme: " + n + " satır");
9 PreparedStatement ps = conn.prepareStatement(
10     "SELECT ad FROM kisiler WHERE ad = ?");
11 ps.setString(1, girdi);
12 n = 0;
13 try (ResultSet rs = ps.executeQuery()) { while
    (rs.next()) n++; }
14 System.out.println("PreparedStatement: " + n + "
    satır");
```

veri tabanına giden SQL:

```
SELECT ad FROM kisiler
WHERE ad = [x' OR '1'='1]
```

id	ad
1	Ali
2	Ayşe
3	Can

Girdi bütünüyle **tek bir metin değeri** olarak bağlanır; içindeki tırnaklar sıradan karakterdir.

Adım 8 / 10

DEĞİŞKENLER

1. ? = x' OR '1'='1

ÇIKTI

Birleştirme: 3 satır

SQL Enjeksiyonu Adım Adım

```
1 String girdi = "x' OR '1'='1";
2 String sql = "SELECT ad FROM kisiler WHERE ad = '"
3           + girdi + "'";
4 int n = 0;
5 try (ResultSet rs = stmt.executeQuery(sql)) {
6     while (rs.next()) n++;
7 }
8 System.out.println("Birleştirme: " + n + " satır");
9 PreparedStatement ps = conn.prepareStatement(
10     "SELECT ad FROM kisiler WHERE ad = ?");
11 ps.setString(1, girdi);
12 n = 0;
13 ▶ try (ResultSet rs = ps.executeQuery()) { while
14     (rs.next()) n++; }
15 System.out.println("PreparedStatement: " + n + "
16     satır");
```

veri tabanına giden SQL:

```
SELECT ad FROM kisiler
WHERE ad = [x' OR '1'='1]
```

id	ad
1	Ali
2	Ayşe
0 satır	Can

koşul

ad = "x' OR '1'='1" → yanlış

ad = "x' OR '1'='1" → yanlış

ad = "x' OR '1'='1" → yanlış

Hiçbir kişinin adı bu 12 karakterlik metne eşit değil: satır yok.

Adım 9 / 10

DEĞİŞKENLER

n = 0

ÇIKTI

Birleştirme: 3 satır

SQL Enjeksiyonu Adım Adım

```
1 String girdi = "x' OR '1'='1";
2 String sql = "SELECT ad FROM kisiler WHERE ad = '"
3           + girdi + "'";
4 int n = 0;
5 try (ResultSet rs = stmt.executeQuery(sql)) {
6     while (rs.next()) n++;
7 }
8 System.out.println("Birleştirme: " + n + " satır");
9 PreparedStatement ps = conn.prepareStatement(
10     "SELECT ad FROM kisiler WHERE ad = ?");
11 ps.setString(1, girdi);
12 n = 0;
13 try (ResultSet rs = ps.executeQuery()) { while
    (rs.next()) n++; }
14 ▶ System.out.println("PreparedStatement: " + n + "
    satır");
```

veri tabanına giden SQL:

```
SELECT ad FROM kisiler
WHERE ad = [x' OR '1'='1]
```

id	ad
1	Ali
2	Ayşe
0 satır	Can

koşul

ad = "x' OR '1'='1" → yanlış

ad = "x' OR '1'='1" → yanlış

ad = "x' OR '1'='1" → yanlış

Aynı girdi, iki farklı sonuç. Kullanıcıdan gelen değer SQL'e her zaman ile verilir.

Adım 10 / 10

DEĞİŞKENLER

n = 0

ÇIKTI

```
Birleştirme: 3 satır
PreparedStatement: 0 satır
```

PREPAREDSTATEMENT

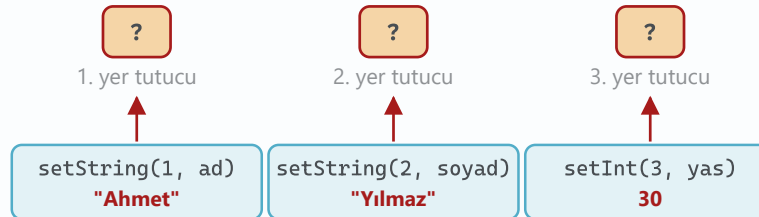
PreparedStatement ve Yer Tutucular

```
static void ekle(Connection conn, String ad, String soyad,
    int yas) throws SQLException {
    String sql = "INSERT INTO kisiler(ad, soyad, yas) "
        + "VALUES(?, ?, ?)";
    try (PreparedStatement ps = conn.prepareStatement(sql)) {
        ps.setString(1, ad);        // 1. soru işareti
        ps.setString(2, soyad);     // 2. soru işareti
        ps.setInt(3, yas);         // 3. soru işareti
        ps.executeUpdate();
    }
}
```

Neden?

- + **Güvenli**: SQL enjeksiyonunu önler.
- + Tırnak sorunu yok: "Ali'nin" sorunsuz kaydedilir.
- + Aynı komut farklı değerlerle **tekrar** kullanılır.

INSERT INTO kisiler(ad, soyad, yas) VALUES(?, ?, ?)



`setXxx(sıra, değer)` her `?` 'i doldurur; sıra **1'den** başlar. Sonra `executeUpdate()` parametresiz çağrılır.

Kişi Programı: Güncelleme ve Silme Metotları

```
static void guncelle(Connection conn, String ad, String soyad,
                    int yas) throws SQLException {
    String sql = "UPDATE kisiler SET soyad = ?, yas = ? "
        + "WHERE ad = ?";
    try (PreparedStatement ps = conn.prepareStatement(sql)) {
        ps.setString(1, soyad);
        ps.setInt(2, yas);
        ps.setString(3, ad);
        ps.executeUpdate();
    }
}

static void sil(Connection conn, String ad) throws SQLException {
    String sql = "DELETE FROM kisiler WHERE ad = ?";
    try (PreparedStatement ps = conn.prepareStatement(sql)) {
        ps.setString(1, ad);
        ps.executeUpdate();
    }
}
```

- Her metot `Connection` 'ı **parametre** olarak alır: bağlantı bir kez açılır, birçok işte kullanılır.
- `?` 'ler SQL'deki **sıraya** göre numaralanır: `guncelle` 'de ad, 3. yer tutucudur.
- `throws SQLException` : hata çağırana iletilir; `main` de `throws` ile bildirdiğinden hata olursa program yığın iziyle durur.

Kişi Programı: Listeleme Metodu

```
static void listele(Connection conn) throws SQLException {  
    String sql = "SELECT id, ad, soyad, yas FROM kisiler";  
    try (Statement st = conn.createStatement();  
        ResultSet rs = st.executeQuery(sql)) {  
        while (rs.next()) {  
            System.out.println(rs.getInt("id") + " " + rs.getString("ad")  
                               + " " + rs.getString("soyad") + " " + rs.getInt("yas"));  
        }  
        System.out.println("---");  
    }  
}
```

ÇIKTI (İLK LİSTELE ÇAĞIRISI)

```
1 Ahmet Akdeniz 35  
2 Ayşe Demir 40  
---
```

- Sorguda değişken değer olmadığından düz `Statement` yeterlidir.
- Her `listele` çağırısı tablonun **o anki** hâlini okur; sonuna `---` yazar.

PREPAREDSTATEMENT

Kişi Programı: Adım Adım

```
1 public static void main(String[] args) throws SQLException {
2     String url = "jdbc:sqlite:kisiler.db";
3     ▶ try (Connection conn = DriverManager.getConnection(url);
4     ▶     Statement stmt = conn.createStatement()) {
5         stmt.execute("CREATE TABLE IF NOT EXISTS kisiler ("
6             + "id INTEGER PRIMARY KEY, ad TEXT NOT NULL, "
7             + "soyad TEXT NOT NULL, yas INTEGER NOT NULL)");
8         ekle(conn, "Ahmet", "Yılmaz", 30);
9         ekle(conn, "Ayşe", "Demir", 40);
10        guncelle(conn, "Ahmet", "Akdeniz", 35);
11        listele(conn);
12        sil(conn, "Ahmet");
13        listele(conn);
14    }
15 }
```

ÇAĞRI YİĞİNİ

main()

kisiler

id	ad	soyad	yas
(tablo boş)			

kisiler.db yoksa oluşturulur;
bağlantı ve stmt açılır.

Adım 1 / 9

ÇIKTI



PREPAREDSTATEMENT

Kişi Programı: Adım Adım

```
1 public static void main(String[] args) throws SQLException {
2     String url = "jdbc:sqlite:kisiler.db";
3     try (Connection conn = DriverManager.getConnection(url);
4         Statement stmt = conn.createStatement()) {
5         ▶ stmt.execute("CREATE TABLE IF NOT EXISTS kisiler ("
6           + "id INTEGER PRIMARY KEY, ad TEXT NOT NULL, "
7           + "soyad TEXT NOT NULL, yas INTEGER NOT NULL)");
8         ekle(conn, "Ahmet", "Yılmaz", 30);
9         ekle(conn, "Ayşe", "Demir", 40);
10        guncelle(conn, "Ahmet", "Akdeniz", 35);
11        listele(conn);
12        sil(conn, "Ahmet");
13        listele(conn);
14    }
15 }
```

ÇAĞRI YİĞİNİ

main()

kisiler

id	ad	soyad	yas
(tablo boş)			

Tablo oluşturulur; henüz boştur.

Adım 2 / 9

ÇIKTI



PREPAREDSTATEMENT

Kişi Programı: Adım Adım

```
1 public static void main(String[] args) throws SQLException {
2     String url = "jdbc:sqlite:kisiler.db";
3     try (Connection conn = DriverManager.getConnection(url);
4         Statement stmt = conn.createStatement()) {
5         stmt.execute("CREATE TABLE IF NOT EXISTS kisiler ("
6             + "id INTEGER PRIMARY KEY, ad TEXT NOT NULL, "
7             + "soyad TEXT NOT NULL, yas INTEGER NOT NULL)");
8         ▶ ekle(conn, "Ahmet", "Yılmaz", 30);
9         ekle(conn, "Ayşe", "Demir", 40);
10        guncelle(conn, "Ahmet", "Akdeniz", 35);
11        listele(conn);
12        sil(conn, "Ahmet");
13        listele(conn);
14    }
15 }
```

ÇAĞRI YİĞİNİ

```
ekle(conn, "Ahmet", "Yılmaz", 30)
```

```
main()
```

kisiler

id	ad	soyad	yas
1	Ahmet	Yılmaz	30

ekle : ? 'ler doldurulur, satır eklenir,
id = 1 .

Adım 3 / 9

ÇIKTI



PREPAREDSTATEMENT

Kişi Programı: Adım Adım

```
1 public static void main(String[] args) throws SQLException {
2     String url = "jdbc:sqlite:kisiler.db";
3     try (Connection conn = DriverManager.getConnection(url);
4         Statement stmt = conn.createStatement()) {
5         stmt.execute("CREATE TABLE IF NOT EXISTS kisiler ("
6             + "id INTEGER PRIMARY KEY, ad TEXT NOT NULL, "
7             + "soyad TEXT NOT NULL, yas INTEGER NOT NULL)");
8         ekle(conn, "Ahmet", "Yılmaz", 30);
9         ► ekle(conn, "Ayşe", "Demir", 40);
10        guncelle(conn, "Ahmet", "Akdeniz", 35);
11        listele(conn);
12        sil(conn, "Ahmet");
13        listele(conn);
14    }
15 }
```

ÇAĞRI YİĞİNİ

```
ekle(conn, "Ayşe", "Demir", 40)
```

```
main()
```

kisiler

id	ad	soyad	yas
1	Ahmet	Yılmaz	30
2	Ayşe	Demir	40

İkinci kişi `id = 2` ile eklenir.

Adım 4 / 9

ÇIKTI



PREPAREDSTATEMENT

Kişi Programı: Adım Adım

```
1 public static void main(String[] args) throws SQLException {
2     String url = "jdbc:sqlite:kisiler.db";
3     try (Connection conn = DriverManager.getConnection(url);
4         Statement stmt = conn.createStatement()) {
5         stmt.execute("CREATE TABLE IF NOT EXISTS kisiler ("
6             + "id INTEGER PRIMARY KEY, ad TEXT NOT NULL, "
7             + "soyad TEXT NOT NULL, yas INTEGER NOT NULL)");
8         ekle(conn, "Ahmet", "Yılmaz", 30);
9         ekle(conn, "Ayşe", "Demir", 40);
10        ▶ guncelle(conn, "Ahmet", "Akdeniz", 35);
11        listele(conn);
12        sil(conn, "Ahmet");
13        listele(conn);
14    }
15 }
```

ÇAĞRI YİĞİNİ

```
guncelle(conn, "Ahmet", "Akdeniz", 35)
```

```
main()
```

kisiler

id	ad	soyad	yas
1	Ahmet	Akdeniz	35
2	Ayşe	Demir	40

WHERE ad = ? : Ahmet'in soyadı ve yaşı değişir.

Adım 5 / 9

ÇIKTI



PREPAREDSTATEMENT

Kişi Programı: Adım Adım

```
1 public static void main(String[] args) throws SQLException {
2     String url = "jdbc:sqlite:kisiler.db";
3     try (Connection conn = DriverManager.getConnection(url);
4         Statement stmt = conn.createStatement()) {
5         stmt.execute("CREATE TABLE IF NOT EXISTS kisiler ("
6             + "id INTEGER PRIMARY KEY, ad TEXT NOT NULL, "
7             + "soyad TEXT NOT NULL, yas INTEGER NOT NULL)");
8         ekle(conn, "Ahmet", "Yılmaz", 30);
9         ekle(conn, "Ayşe", "Demir", 40);
10        guncelle(conn, "Ahmet", "Akdeniz", 35);
11        ► listele(conn);
12        sil(conn, "Ahmet");
13        listele(conn);
14    }
15 }
```

ÇAĞRI YİĞİNİ

listele(conn)

main()

kisiler

id	ad	soyad	yas
1	Ahmet	Akdeniz	35
2	Ayşe	Demir	40

listele tüm satırları ve --- yazar.

Adım 6 / 9

ÇIKTI

```
1 Ahmet Akdeniz 35
2 Ayşe Demir 40
---
```

PREPAREDSTATEMENT

Kişi Programı: Adım Adım

```
1 public static void main(String[] args) throws SQLException {
2     String url = "jdbc:sqlite:kisiler.db";
3     try (Connection conn = DriverManager.getConnection(url);
4         Statement stmt = conn.createStatement()) {
5         stmt.execute("CREATE TABLE IF NOT EXISTS kisiler ("
6             + "id INTEGER PRIMARY KEY, ad TEXT NOT NULL, "
7             + "soyad TEXT NOT NULL, yas INTEGER NOT NULL)");
8         ekle(conn, "Ahmet", "Yılmaz", 30);
9         ekle(conn, "Ayşe", "Demir", 40);
10        guncelle(conn, "Ahmet", "Akdeniz", 35);
11        listele(conn);
12        ► sil(conn, "Ahmet");
13        listele(conn);
14    }
15 }
```

ÇAĞRI YİĞİNİ

```
sil(conn, "Ahmet")
```

```
main()
```

kisiler

id	ad	soyad	yas
1	Ahmet	Akdeniz	35
2	Ayşe	Demir	40

Ahmet'in satırı silinir.

Adım 7 / 9

ÇIKTI

```
1 Ahmet Akdeniz 35
2 Ayşe Demir 40
---
```

PREPAREDSTATEMENT

Kişi Programı: Adım Adım

```
1 public static void main(String[] args) throws SQLException {
2     String url = "jdbc:sqlite:kisiler.db";
3     try (Connection conn = DriverManager.getConnection(url);
4         Statement stmt = conn.createStatement()) {
5         stmt.execute("CREATE TABLE IF NOT EXISTS kisiler ("
6             + "id INTEGER PRIMARY KEY, ad TEXT NOT NULL, "
7             + "soyad TEXT NOT NULL, yas INTEGER NOT NULL)");
8         ekle(conn, "Ahmet", "Yılmaz", 30);
9         ekle(conn, "Ayşe", "Demir", 40);
10        guncelle(conn, "Ahmet", "Akdeniz", 35);
11        listele(conn);
12        sil(conn, "Ahmet");
13        ▶ listele(conn);
14    }
15 }
```

ÇAĞRI YİĞİNİ

listele(conn)

main()

kisiler

id	ad	soyad	yas
2	Ayşe	Demir	40

Tabloda yalnızca Ayşe kaldı.

Adım 8 / 9

ÇIKTI

```
1 Ahmet Akdeniz 35
2 Ayşe Demir 40
---
2 Ayşe Demir 40
---
```

PREPAREDSTATEMENT

Kişi Programı: Adım Adım

```
1 public static void main(String[] args) throws SQLException {
2     String url = "jdbc:sqlite:kisiler.db";
3     try (Connection conn = DriverManager.getConnection(url);
4         Statement stmt = conn.createStatement()) {
5         stmt.execute("CREATE TABLE IF NOT EXISTS kisiler ("
6             + "id INTEGER PRIMARY KEY, ad TEXT NOT NULL, "
7             + "soyad TEXT NOT NULL, yas INTEGER NOT NULL)");
8         ekle(conn, "Ahmet", "Yılmaz", 30);
9         ekle(conn, "Ayşe", "Demir", 40);
10        guncelle(conn, "Ahmet", "Akdeniz", 35);
11        listele(conn);
12        sil(conn, "Ahmet");
13        listele(conn);
14    }
15 }
```

ÇAĞRI YİĞİNİ

main()

kisiler

id	ad	soyad	yas
2	Ayşe	Demir	40

Bağlantı kapanır; veri `kisiler.db` 'de kalıcıdır.

Adım 9 / 9

ÇIKTI

```
1 Ahmet Akdeniz 35
2 Ayşe Demir 40
---
2 Ayşe Demir 40
---
```

Aynı Kod, Başka Veri Tabanı: MySQL

```
String url = "jdbc:mysql://localhost:3306/testdb";
String kullanıcı = "root";
String sifre = "sifreniz";
String sql = "SELECT id, name, email FROM users";
try (Connection conn =
    DriverManager.getConnection(url, kullanıcı, sifre);
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery(sql)) {
    while (rs.next()) {
        System.out.println(rs.getInt("id") + " "
            + rs.getString("name"));
    }
} catch (SQLException e) {
    System.out.println(e.getMessage());
}
```

- JDBC kodu **aynıdır**; değişenler: **adres, kullanıcı adı, şifre** ve sürücü jar'ı (`mysql-connector-j`).
- MySQL bir **sunucudur**: önce kurulup çalıştırılmalı, `testdb` veri tabanı oluşturulmalıdır.
- Sürücü sınıfı `com.mysql.cj.jdbc.Driver` 'dır; eski `com.mysql.jdbc.Driver` adı kullanımdan kalkmıştır.
- Gerçek projelerde şifre koda yazılmaz; ayar dosyasından veya ortam değişkeninden okunur.

İŞLEMLER

İşlem (Transaction) ve ACID

- **Havale:** Ece'nin hesabına 300 TL ekle **ve** Ali'nin hesabından 300 TL düş. İki `UPDATE` 'ten biri olur diğeri olmazsa para **yoktan var olur** ya da **kaybolur**.
- **İşlem** (transaction): tek bir iş birimi olarak yürütülen bir veya daha fazla SQL komutu: **ya hepsi ya hiçbiri**.
- JDBC'de varsayılan **auto-commit** açıktır: her komut hemen kalıcı olur.
- `conn.setAutoCommit(false)` → komutlar → `conn.commit()` (kalıcı yap) veya `conn.rollback()` (geri al).

Atomicity

Bölünmezlik: işlemin ya tamamı ya hiçbiri uygulanır.

Consistency

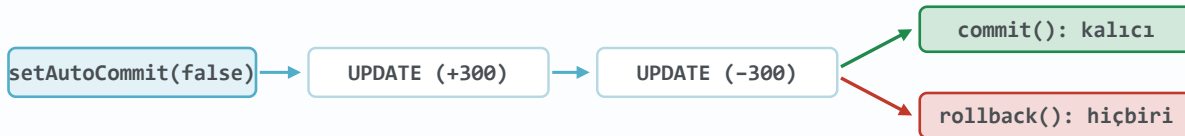
Tutarlılık: işlem sonunda tüm kısıtlar (ör. bakiye ≥ 0) sağlanır.

Isolation

Yalıtım: eş zamanlı işlemler birbirinin yarım hâlini görmez.

Durability

Kalıcılık: `commit` edilen veri, elektrik kesilse de kaybolmaz.



İŞLEMLER

Havale Örneği: Kurulum ve Sonuç

```
static final String YATIR =
    "UPDATE hesaplar SET bakiye = bakiye + ? WHERE id = ?";
static final String CEK =
    "UPDATE hesaplar SET bakiye = bakiye - ? WHERE id = ?";

public static void main(String[] args) throws SQLException {
    String url = "jdbc:sqlite::memory:"; // bellekte, geçici
    try (Connection conn = DriverManager.getConnection(url);
        Statement st = conn.createStatement()) {
        st.execute("CREATE TABLE hesaplar (id INTEGER PRIMARY KEY, "
            + "sahip TEXT, bakiye INTEGER CHECK (bakiye >= 0))");
        st.execute("INSERT INTO hesaplar(sahip, bakiye) "
            + "VALUES ('Ali', 1000), ('Ece', 500)");
        havale(conn, 1, 2, 300); // Ali → Ece 300 TL
        havale(conn, 1, 2, 5000); // Ali'nin parası yetmez
        yazdir(conn);
    }
}
```

ÇIKTI

300 TL aktarıldı.
Geri alındı: 5000 TL
Ali: 700
Ece: 800

- **CHECK (bakiye >= 0)** : bakiyeyi eksiye düşüren komut **SQLException** fırlatır.
- İkinci havalede Ece'ye 5000 TL **yatırılmıştı**; **rollback** bunu da geri aldı.
- **yazdir** , **listele** gibi bir **SELECT** döngüsüdür.

İŞLEMLER

commit ve rollback: Adım Adım

```
1 static void havale(Connection conn, int kimden, int kime,  
2 int tutar) throws SQLException {  
3     conn.setAutoCommit(false); // işlem başlar  
4     try (PreparedStatement p1 = conn.prepareStatement(YATIR);  
5         PreparedStatement p2 = conn.prepareStatement(CEK)) {  
6         p1.setInt(1, tutar); p1.setInt(2, kime);  
7         p1.executeUpdate(); // alıcıya yatır  
8         p2.setInt(1, tutar); p2.setInt(2, kimden);  
9         p2.executeUpdate(); // göndericiden çek  
10        conn.commit(); // kalıcı yap  
11        System.out.println(tutar + " TL aktarıldı.");  
12    } catch (SQLException e) {  
13        conn.rollback(); // geri al  
14        System.out.println("Geri alındı: " + tutar + " TL");  
15    } finally {  
16        conn.setAutoCommit(true);  
17    }  
18 }
```

ÇAĞRI YİĞİNİ

```
havale(conn, 1, 2, 300)
```

```
main()
```

işlem başladı

id	sahip	bakiye
1	Ali	1000
2	Ece	500

Auto-commit kapatılır: bundan sonraki değişiklikler **commit** 'e kadar **geçicidir**.

Adım 1 / 8

ÇIKTI



İŞLEMLER

commit ve rollback: Adım Adım

```
1 static void havale(Connection conn, int kimden, int kime,  
2                     int tutar) throws SQLException {  
3     conn.setAutoCommit(false); // işlem başlar  
4     try (PreparedStatement p1 = conn.prepareStatement(YATIR);  
5         PreparedStatement p2 = conn.prepareStatement(CEK)) {  
6         ▶ p1.setInt(1, tutar); p1.setInt(2, kime);  
7         ▶ p1.executeUpdate(); // alıcıya yatır  
8         p2.setInt(1, tutar); p2.setInt(2, kimden);  
9         p2.executeUpdate(); // göndericiden çek  
10        conn.commit(); // kalıcı yap  
11        System.out.println(tutar + " TL aktarıldı.");  
12    } catch (SQLException e) {  
13        conn.rollback(); // geri al  
14        System.out.println("Geri alındı: " + tutar + " TL");  
15    } finally {  
16        conn.setAutoCommit(true);  
17    }  
18 }
```

ÇAĞRI YİĞİNİ

```
havale(conn, 1, 2, 300)
```

```
main()
```

geçici (commit yok)

id	sahip	bakiye
1	Ali	1000
2	Ece	800

Ece'ye 300 TL yatırılır: 500 → 800.

Adım 2 / 8

ÇIKTI



İŞLEMLER

commit ve rollback: Adım Adım

```
1 static void havale(Connection conn, int kimden, int kime,
2                     int tutar) throws SQLException {
3     conn.setAutoCommit(false); // işlem başlar
4     try (PreparedStatement p1 = conn.prepareStatement(YATIR);
5         PreparedStatement p2 = conn.prepareStatement(CEK)) {
6         p1.setInt(1, tutar); p1.setInt(2, kime);
7         p1.executeUpdate(); // alıcıya yatır
8         ▶ p2.setInt(1, tutar); p2.setInt(2, kimden);
9         ▶ p2.executeUpdate(); // göndericiden çek
10        conn.commit(); // kalıcı yap
11        System.out.println(tutar + " TL aktarıldı.");
12    } catch (SQLException e) {
13        conn.rollback(); // geri al
14        System.out.println("Geri alındı: " + tutar + " TL");
15    } finally {
16        conn.setAutoCommit(true);
17    }
18 }
```

ÇAĞRI YİĞİNİ

```
havale(conn, 1, 2, 300)
```

```
main()
```

geçici (commit yok)

id	sahip	bakiye
1	Ali	700
2	Ece	800

Ali'den 300 TL çekilir: 1000 → 700. Kısıt sağlanıyor.

Adım 3 / 8

ÇIKTI



İŞLEMLER

commit ve rollback: Adım Adım

```
1 static void havale(Connection conn, int kimden, int kime,  
2 int tutar) throws SQLException {  
3     conn.setAutoCommit(false); // işlem başlar  
4     try (PreparedStatement p1 = conn.prepareStatement(YATIR);  
5         PreparedStatement p2 = conn.prepareStatement(CEK)) {  
6         p1.setInt(1, tutar); p1.setInt(2, kime);  
7         p1.executeUpdate(); // alıcıya yatır  
8         p2.setInt(1, tutar); p2.setInt(2, kimden);  
9         p2.executeUpdate(); // göndericiden çek  
10        conn.commit(); // kalıcı yap  
11        System.out.println(tutar + " TL aktarıldı.");  
12    } catch (SQLException e) {  
13        conn.rollback(); // geri al  
14        System.out.println("Geri alındı: " + tutar + " TL");  
15    } finally {  
16        conn.setAutoCommit(true);  
17    }  
18 }
```

ÇAĞRI YİĞİNİ

```
havale(conn, 1, 2, 300)
```

```
main()
```

kalıcı (commit edildi)

id	sahip	bakiye
1	Ali	700
2	Ece	800

`commit()` : iki değişiklik birlikte kalıcı olur.

Adım 4 / 8

ÇIKTI

```
300 TL aktarıldı.
```

İŞLEMLER

commit ve rollback: Adım Adım

```
1 static void havale(Connection conn, int kimden, int kime,  
2                     int tutar) throws SQLException {  
3     conn.setAutoCommit(false);           // işlem başlar  
4     try (PreparedStatement p1 = conn.prepareStatement(YATIR);  
5         PreparedStatement p2 = conn.prepareStatement(CEK)) {  
6         p1.setInt(1, tutar); p1.setInt(2, kime);  
7         p1.executeUpdate();              // alıcıya yatır  
8         p2.setInt(1, tutar); p2.setInt(2, kimden);  
9         p2.executeUpdate();              // göndericiden çek  
10        conn.commit();                   // kalıcı yap  
11        System.out.println(tutar + " TL aktarıldı.");  
12    } catch (SQLException e) {  
13        conn.rollback();                  // geri al  
14        System.out.println("Geri alındı: " + tutar + " TL");  
15    } finally {  
16        ▶ conn.setAutoCommit(true);  
17    }  
18 }
```

ÇAĞRI YİĞİNİ

```
havale(conn, 1, 2, 300)
```

```
main()
```

kalıcı durum

id	sahip	bakiye
1	Ali	700
2	Ece	800

finally : auto-commit yeniden açılır;
metot biter.

Adım 5 / 8

ÇIKTI

```
300 TL aktarıldı.
```

İŞLEMLER

commit ve rollback: Adım Adım

```
1 static void havale(Connection conn, int kimden, int kime,  
2 int tutar) throws SQLException {  
▶ conn.setAutoCommit(false); // işlem başlar  
4 try (PreparedStatement p1 = conn.prepareStatement(YATIR);  
5 PreparedStatement p2 = conn.prepareStatement(CEK)) {  
▶ p1.setInt(1, tutar); p1.setInt(2, kime);  
▶ p1.executeUpdate(); // alıcıya yatır  
8 p2.setInt(1, tutar); p2.setInt(2, kimden);  
9 p2.executeUpdate(); // göndericiden çek  
10 conn.commit(); // kalıcı yap  
11 System.out.println(tutar + " TL aktarıldı.");  
12 } catch (SQLException e) {  
13 conn.rollback(); // geri al  
14 System.out.println("Geri alındı: " + tutar + " TL");  
15 } finally {  
16 conn.setAutoCommit(true);  
17 }  
18 }
```

ÇAĞRI YİĞİNİ

```
havale(conn, 1, 2, 5000)
```

```
main()
```

geçici (commit yok)

id	sahip	bakiye
1	Ali	700
2	Ece	5800

İkinci çağrı: Ece'ye 5000 TL yatırılır (geçici):
800 → 5800.

Adım 6 / 8

ÇIKTI

```
300 TL aktarıldı.
```

İŞLEMLER

commit ve rollback: Adım Adım

```
1 static void havale(Connection conn, int kimden, int kime,  
2 int tutar) throws SQLException {  
3     conn.setAutoCommit(false); // işlem başlar  
4     try (PreparedStatement p1 = conn.prepareStatement(YATIR);  
5         PreparedStatement p2 = conn.prepareStatement(CEK)) {  
6         p1.setInt(1, tutar); p1.setInt(2, kime);  
7         p1.executeUpdate(); // alıcıya yatır  
8         ▶ p2.setInt(1, tutar); p2.setInt(2, kimden);  
9         ▶ p2.executeUpdate(); // göndericiden çek  
10        conn.commit(); // kalıcı yap  
11        System.out.println(tutar + " TL aktarıldı.");  
12    } catch (SQLException e) {  
13        conn.rollback(); // geri al  
14        System.out.println("Geri alındı: " + tutar + " TL");  
15    } finally {  
16        conn.setAutoCommit(true);  
17    }  
18 }
```

ÇAĞRI YİĞİNİ

```
havale(conn, 1, 2, 5000)
```

```
main()
```

geçici (commit yok)

id	sahip	bakiye
1	Ali	700
2	Ece	5800

700 – 5000 < 0 olurdu: **CHECK** bozulur,
SQLException fırlar.

Adım 7 / 8

ÇIKTI

```
300 TL aktarıldı.
```

İŞLEMLER

commit ve rollback: Adım Adım

```
1 static void havale(Connection conn, int kimden, int kime,
2                     int tutar) throws SQLException {
3     conn.setAutoCommit(false); // işlem başlar
4     try (PreparedStatement p1 = conn.prepareStatement(YATIR);
5          PreparedStatement p2 = conn.prepareStatement(CEK)) {
6         p1.setInt(1, tutar); p1.setInt(2, kime);
7         p1.executeUpdate(); // alıcıya yatır
8         p2.setInt(1, tutar); p2.setInt(2, kimden);
9         p2.executeUpdate(); // göndericiden çek
10        conn.commit(); // kalıcı yap
11        System.out.println(tutar + " TL aktarıldı.");
12    } catch (SQLException e) {
13        ▶ conn.rollback(); // geri al
14        ▶ System.out.println("Geri alındı: " + tutar + " TL");
15    } finally {
16        conn.setAutoCommit(true);
17    }
18 }
```

ÇAĞRI YİĞİNİ

```
havale(conn, 1, 2, 5000)
```

```
main()
```

rollback: geri alındı

id	sahip	bakiye
1	Ali	700
2	Ece	800

rollback() : geçici yatırma da **geri alınır**.
Para ne kayboldu ne çoğaldı.

Adım 8 / 8

ÇIKTI

```
300 TL aktarıldı.
Geri alındı: 5000 TL
```

ÖZET

Özet

Konu	Ana fikir
Veri tabanı	Kalıcı, düzenli, sorgulanabilir veri; VTYS yönetir. Tablo, satır, sütun, birincil anahtar.
SQLite	Sunucusuz, gömülü, tek dosya. Adres: <code>jdbc:sqlite:sample.db</code> ; sürücü jar'ı classpath'te olmalı.
SQL	<code>CREATE TABLE</code> , <code>INSERT</code> , <code>SELECT</code> , <code>UPDATE</code> , <code>DELETE</code> ; <code>WHERE</code> , <code>ORDER BY</code> , <code>LIMIT</code> .
JDBC	<code>DriverManager</code> → <code>Connection</code> → <code>Statement</code> / <code>PreparedStatement</code> → <code>ResultSet</code> .
Çalıştırma	<code>execute</code> (tablo), <code>executeUpdate</code> → <code>int</code> , <code>executeQuery</code> → <code>ResultSet</code> ; <code>while</code> (<code>rs.next()</code>) .
Güvenlik	Kullanıcı girdisi her zaman <code>?</code> ve <code>setXxx</code> ile; SQL metnine eklenmez.
İşlem	<code>setAutoCommit(false)</code> , <code>commit()</code> , <code>rollback()</code> : ya hepsi ya hiçbiri (ACID).
Kaynaklar	try-with-resources: <code>rs</code> , <code>stmt</code> , <code>conn</code> ters sırada kapanır; <code>SQLException</code> kontrol edilen istisnadır.

ÖZET

Sık Yapılan Hatalar

Sürücü classpath'te değil

Jar eklenmemiş:

```
No suitable driver found for  
jdbc:sqlite:sample.db
```

SQLException'ı yakalamamak

try veya throws yok (derleme hatası):

```
unreported exception SQLException; must be  
caught or declared to be thrown
```

Tablo yok veya adı yanlış

Tablo oluşturulmadan sorgu:

```
no such table: users
```

Metni tırnaksız yazmak

```
VALUES(Ahmet, ...):
```

```
no such column: Ahmet
```

Kesme işaretli metni birleştirmek

```
VALUES('Ali'nin'):
```

```
near "nin": syntax error
```

Yanlış execute metodu

```
executeQuery ile INSERT:
```

```
query does not return ResultSet
```

İndisi 0'dan başlatmak

```
rs.getString(0) ; sıra 1'den başlar:
```

```
column 0 out of bounds [1,3]
```

WHERE'siz UPDATE / DELETE

Hata vermez; **tüm satırları** değiştirir ya da siler.

ÖZET

Örnek Problemler 1

Problem 1: SELECT hangi satırları döndürür?

```
CREATE TABLE users (id INTEGER PRIMARY KEY, name TEXT);
INSERT INTO users(name) VALUES ('Ali');
INSERT INTO users(name) VALUES ('Can');
DELETE FROM users WHERE id = 2;
INSERT INTO users(name) VALUES ('Ece');
UPDATE users SET name = 'Efe' WHERE id = 1;
SELECT * FROM users;
```

Problem 2

Kişi programı (`kisiler.db` ile) **ikinci kez** çalıştırılıyor. Çıktının ilk bloğu (ilk `---` 'e kadar) ve son bloğu nedir?

► Cevapları göster

ÖZET

Örnek Problemler 2

Problem 3: Giriş kontrolü

```
String sql = "SELECT * FROM kisiler WHERE ad = '" + ad  
            + "' AND sifre = '" + sifre + "'";  
ResultSet rs = stmt.executeQuery(sql);  
boolean girisBasarili = rs.next();
```

Tabloda 3 kişi var. Kullanıcı ad olarak `ahmet`, şifre olarak `'`
`OR '1'='1` yazarsa oluşan SQL nedir, kaç satır döner, giriş
başarılı olur mu?

► Cevapları göster

Problem 4

`havale` metodundan
`setAutoCommit(false)`, `commit()` ve
`rollback()` satırları çıkarılsaydı (her komut
kendi başına kalıcı) program sonunda Ali ve
Ece'nin bakiyesi ne olurdu?

ÖZET

Alıştırmalar

1. Silme programını değiştirin: `executeUpdate` 0 döndürürse `Kayıt bulunamadı.` yazsın.
2. `ogrenciler(id, ad, vize, final)` tablosunu oluşturun; Ada (60, 70), Bora (30, 50), Cem (80, 40) öğrencilerini `PreparedStatement` ile bir döngüde ekleyin. Ortalaması (vize %40, final %60) 50 ve üzeri olanları büyükten küçüğe listeleyin.
3. Menülü bir program yazın: 1-Ekle, 2-Güncelle, 3-Sil, 4-Liste. `Scanner` ve `switch` kullanın; güncellenecek/silinecek id yoksa `Kullanıcı bulunamadı.` yazsın.
4. Bir oyunun skorlarını `skorlar(oyuncu, puan)` tablosuna kaydedin ve en yüksek 10 skoru listeleyin.

► Cevaplar / ipuçları

Kaynaklar

- Oracle, *The Java Tutorials: JDBC Database Access* (Establishing a Connection; Processing SQL Statements with JDBC; Using Prepared Statements; Using Transactions).
- Oracle, *Java SE 21 API Documentation*: `java.sql` paketi (`DriverManager` , `Connection` , `PreparedStatement` , `ResultSet`).
- SQLite belgeleri, sqlite.org: *SQL As Understood By SQLite; Datatypes In SQLite; Appropriate Uses For SQLite*.
- Xerial, *SQLite JDBC Driver*, github.com/xerial/sqlite-jdbc.
- P. Deitel ve H. Deitel, *Java: How to Program*, 11. baskı, Pearson (Accessing Databases with JDBC).
- C. S. Horstmann, *Core Java, Volume II: Advanced Features*, 12. baskı, Pearson (Database Programming).
- OWASP, *SQL Injection Prevention Cheat Sheet*.

Son

Bölüm 10: Veri Tabanı

Sonraki bölüm: Bölüm 11: Test ve Hata Ayıklama · [Terim sözlüğü](#)