YILDIZ TEKNİK ÜNİVERSİTESİ

SQL Programlama Teknikleri

Öğr. Gör. Dr. Yasemin Topuz

Yıldız Teknik Üniversitesi

BLM3041 – Veritabanı Yönetimi

07 Kasım 2025 | YTÜ Bilgisayar Mühendisliği

Neler konuşacağız?

- Veritabanı Programlama Yaklaşımları
  - Embedded (Gömülü) SQL
  - Veritabanı İşlev Kütüphane, JDBC (API)
  - Stored Prosedure, SQL/PSM

Java, C, Python vb. genel amaçlı dillerdeki programlar aracılığıyla bir veritabanına erişim ve veritabanlarında değişiklik yapmak için çeşitli teknikler.

Veritabanı Programlama

Amaç

- Bir veritabanına bir uygulama programından erişmek ve veri işlemlerini kodla yönetmek (SQL konsolu, pgAdmin vb. etkileşimli arayüzlerin yerine)

Neden?

- Etkileşimli arayüzler pratiktir ama yeterli değildir: otomasyon, tekrar edilebilirlik ve entegrasyon sınırlıdır.
  - Her gece 02:00'de rapor üretip yöneticilere e-posta atmak

- Günümüzde veritabanı işlemlerinin büyük çoğunluğu uygulama programları (özellikle web ve mobil uygulamalar) üzerinden yürütülür.

Veritabanı Programlama

- »Veritabanı programlama» dediğimizde, SQL'i bir GUI'de yazmaktan ziyade kod içinden SQL çalıştırmayı (JDBC/ODBC, psycopg2, SQLAlchemy, PDO, .NET SqlClient vb.) kastediyoruz.

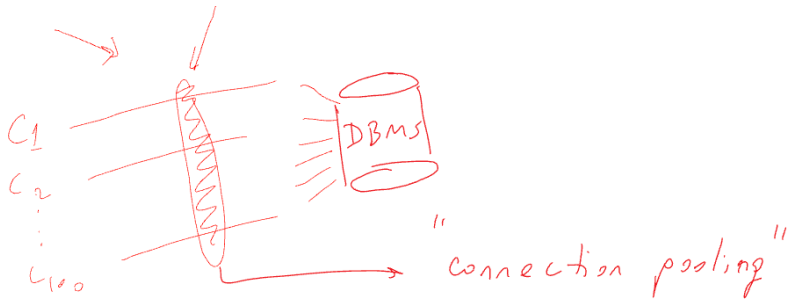
Neden gerekli?

- İş mantığı + veri aynı akışta birleşir (API'ler, arka uç servisler).
- **Otomasyon:** zamanlanmış işler, arka plan kuyrukları.
- **Güvenlik ve denetim:** parametrik sorgular, rol tabanlı erişim, loglama.
- **Performans:** bağlantı havuzu, önbellekleme, toplu işlemler.

Etkileşimli araçlar öğrenme/diagnostik için mükemmel; ancak üretimde SQL programlama erişim standarttır.

Veritabanı Programlamada Temel Adımlar

1) **İstemci program veritabanı sunucusuna bir bağlantı açar.**
   - Sürücü/istemci kütüphanesi yüklenir (JDBC, psycopg2, Npgsql, PDO, SqlClient…).
   - Kimlik doğrulama + TLS/SSL, zaman aşımı ve pool ayarları yapılır.
2) **İstemci program sorguları gönderir ve/veya veriyi günceller.**
   - İşlemleri (transaction) mantıksal bütünlük içinde grupla:
     BEGIN → (SELECT/INSERT/UPDATE/DELETE) → COMMIT/ROLLBACK.
   - Hata durumlarını yakala, tekrar dene/geri al.
   - Parametrik sorgular kullan (SQL injection'a karşı).
3) **Erişim bittiğinde bağlantı kapatılır (veya havuza iade edilir).**
   - Bağlantıyı açık bırakmak kaynak tüketir.
   - DBMS default olarak aynı anda maximum connection pool size değeri vardır.



Veritabanı Programlamada Temel Adımlar

Yaygın hatalar ve kaçınma

- Kimlik bilgilerini koda gömmek (Parametre yerine string birleştirme) → sızıntı riski.
- *sslmode=disable* ile prod'a çıkmak → şifrelenmemiş trafik.
- Çok büyük max pool size → DB tarafında oturum şişmesi ve kilitlenmeler.
- Zaman aşımı yok (Transaction'ı uzun tutmak)→ iş parçacıkları "asılı" kalır, sistem tıkanır.

Özetle: kimlik doğrulama kim olduğunuzu kanıtlar, TLS/SSL konuşmayı güvenli yapar, timeout'lar takılmaları önler, pool ise performans/ölçek sağlar.

Bu dördü üretim-kalitesinde veritabanı bağlantısının temel taşıdır.

Veritabanı Programlamada Temel Adımlar

Veritabanı Programlamada Temel Sorun: **“Impedance Mismatch”**

- **Empedans uyumsuzluğu:** Programlama dili (Java, C/C#, Python vb.) ile ilişkisel/SQL veri modeli arasındaki uyuşmazlıklar.
  - İlişkisel/SQL dünyasının küme-tabanlı (SET) şema-odaklı modeli ile uygulama kodunun nesne/tekil-kayıt dünyasını barıştırma problemidir !
- **Tür uyumsuzluğu:** Her programlama dili için yeni bir bağlama gerektirir. (**CASTING**)
- **SET-öncelikli SQL ve CURSOR**
- SQL küme (SET/BAG) odaklı düşünür; “tek sorgu → çok satır”
  Çoğu programlama dili döngüyle “tek tek satır” (record-at-a-time) işleme eğilimindedir.
  - Sorgu sonuçları üzerinde döngü oluşturmak ve tek tek değerleri işlemek için özel yineleyiciler olan **CURSORS** kullanılır.
  - **CURSOR:** Sorgu sonucunu satır satır ilerleten imleç

Uygulama Programları: Yaklaşımlar

Bir uygulama veritabanıyla konuşmak için üç ana yaklaşım vardır:

1) **Embedded SQL:** SQL'i koda göm ve derlet (eski tarz),

2) **Veritabanı İşlev Kütüphanesi:** Sürücülü API ile konuş (modern tarz, JDBC gibi),

3) **Stored Procedure:** İşi direkt veritabanının içine göm.

Uygulama Programları: Yaklaşımlar

1) **Gömülü Komutlar (Embedded SQL, SQLJ)**
   - **Tanım:** SQL ifadelerinin yüksek seviyeli dillere (C, COBOL, Java/SQLJ) doğrudan gömülmesi.
   - **Nasıl çalışır?** Kaynak koda EXEC SQL ... / #sql { ... } blokları yazılır; preprocessor/translator bu blokları sürücü çağrılarına dönüştürür.
   - **Derleme zinciri:** precompile → compile → link. SQL sözdizimi hatalarının bir bölümü derleme zamanında yakalanabilir.
   - **Taşınabilirlik:** Platform/DBMS'nin dil bağımlılığı yüksektir; farklı ortamlarda yeniden derleme gerekir.

   Artı: Bazı hataları derleme aşamasında yakalar; sabit (statik) SQL senaryolarında performanslı olabilir.

   Eksi: Taşınabilirlik düşüktür (farklı platformda yeniden derleme), derleme/dağıtım zinciri karmaşıktır, diller arası geçiş (ör. C → Java) genelde yeniden yazım gerektirir.

Uygulama Programları: Yaklaşımlar

2) **Veritabanı İşlev Kütüphanesi — JDBC/ODBC, ADO.NET, PDO, psycopg2…**

- **Tanım:** Uygulamanın kendi dilinde (Java, C#, Python vb.) veritabanı API'leri/sürücüleri (JDBC, ODBC, ADO.NET, PDO, psycopg2…) ile SQL çalıştırma yaklaşımı.
- **Nasıl çalışır?** Bağlanmak için server-client modeli; connection alınır → prepared statement ile komutlar/parametreler gönderilir → result set okunur → bağlantı kapatılır/havuza iade edilir. Transaction, timeout, pooling vb. API üzerinden yönetilir.
- **Derleme zinciri:** Preprocessor yok. Normal derleme; sürücü JAR/nuget/pip paketi eklenir, çalışma zamanında yüklenir.
- **Taşınabilirlik:** Yüksek. Sürücüyü değiştirerek farklı DBMS'lere bağlanılabilir.

Artı: Çalışma zamanında esnek ve hızlı; parametreli sorgular (güvenli), connection pooling, çoklu veritabanına erişim.

Eksi: SQL sözdizimi/şema hataları çoğunlukla run-time'da görünür. Dinamik sonuç şeması için ek metadata/descriptor kodu gerekebilir.

Uygulama Programları: Yaklaşımlar

3) **Stored Procedure/Tetikleyici Dilleri**

- **Tanım:** DB-yerel dillerle (PL/pgSQL, PL/SQL, T-SQL vb.) yazılan stored procedure / function / trigger'lar veritabanında kalıcı olarak saklanır ve sunucuda çalışır.
- **Nasıl çalışır?** Uygulama sadece CALL/SELECT ile prosedürü tetikler; işlem mantığı DB içinde, veriye en yakın yerde yürür (set-tabanlı optimizasyon, ağ turu yok).
- **Derleme zinciri:** DBMS'in kendi derleyicisi/yorumlayıcısı kullanılır (CREATE PROCEDURE/FUNCTION). Uygulama tarafında ek preprocessor yok
- **Taşınabilirlik:** Düşük–orta. Sözdizimi ve özellikler DBMS'e bağımlıdır (PL/pgSQL ≠ PL/SQL ≠ T-SQL). Taşıma genelde yeniden yazım ister.

Artı: En hızlı veri işleme (minimum network), atomiklik ve güvenlik (tek transaction bloğu, tablo yerine prosedür yetkisi), Impedans uyumsuzluğunu azaltır.

Eksi: Sınırlı esneklik (büyük mimarilerde iş mantığını DB'ye kilitler). PL/pgSQL ≠ PL/SQL ≠ T-SQL. Bir DBMS'ten diğerine geçmek çoğu zaman yeniden yazım ister.

Embedded SQL

- Çoğu SQL ifadesi, C, Java, Python gibi genel amaçlı bir ana programlama diline gömülebilir.
- VTYS'ye özgü "ön işlemci", SQL'i ana dilde fonksiyon çağrılarına dönüştürür.
- Gömülü bir SQL ifadesi, ana dil ifadelerinden, EXEC SQL veya EXEC SQL BEGIN ve eşleşen bir END-EXEC veya EXEC SQL END (veya noktalı virgül) arasına yerleştirilmesiyle ayrılır.
  - Sözdizimi dile göre değişebilir.
  - Her iki dil tarafından paylaşılan değişkenler SQL tarafında :degisken şeklinde iki nokta ile kullanılır; ana programda iki noktasız normal değişkendir.
  - **Derleme aşaması:** precompile → compile → link; hedef platforma göre yeniden derleme gerekebilir.

Embedded SQL - Değişken Bildirimi (Variable Declaration)

Temel İlke

- «DECLARE» SECTION içinde bildirilen değişkenler paylaşılan değişkenlerdir; SQL içinde iki nokta (:) önekiyle kullanılır, ana programda normal değişkendir.
- **SQLCODE**, veritabanı ile program arasında hataları/istisnaları iletmek için kullanılır.

```
int loop;
EXEC SQL BEGIN DECLARE SECTION;
    varchar dname[16], fname[16], lname[16], …;
    char ssn[10], bdate[11], …;
    float salary, raise;
    int dno, dnumber, SQLCODE, …;
EXEC SQL END DECLARE SECTION;
```

Embedded SQL - Veritabanına Bağlanma (Connecting to a Database)

Çoklu Bağlantı

- Bir program birden çok bağlantı açabilir; çoğu sistemde aynı anda yalnızca biri etkin bağlamdır.

Bağlanma / Kimlik Doğrulama (şematik dil)

```
CONNECT TO servername AS connectionname
AUTHORIZATION user_account_info;
```

Etkin Bağlantıyı Değiştirme

```
SET CONNECTION connectionname;
```

Bağlantıyı Sonlandırma

```
DISCONNECT connectionname;      -- veya DISCONNECT ALL;
```

Embedded SQL - Örnek 1: Tek bir tuple'ı alma (retrieving single tuple)

```
loop = 1;
while (loop) {
        prompt (“Enter SSN: “, ssn);
        EXEC SQL
                SELECT FNAME, LNAME, ADDRESS, SALARY
                INTO :fname, :lname, :address, :salary
                FROM EMPLOYEE where SSN = :ssn;

                if (SQLCODE == 0) printf(fname, …);
                else printf(“SSN does not exist: “, ssn);
                prompt("More SSN? (1=yes, 0=no): ", loop);
        END-EXEC
}
```

Amaç

- Kullanıcıdan SSN (TC benzeri kimlik) alıp EMPLOYEE tablosundan tek satır okumak.
- Sonucu paylaşılan değişkenlere (:fname, :lname, …) aktarmak.
- Başarı/başarısızlık durumunu SQLCODE ile kontrol etmek.

Akış

- 1) Girdi al → 2) SELECT … INTO (tek satır beklenir) → 3) SQLCODE kontrolü → 4) Yazdır veya “kayıt yok” mesajı → 5) Devam/çıkış.

Embedded SQL - Örnek 2: Birden Çok Kayıt Getirme (retrieving multiple tuples)

Ne zaman CURSOR?

- Sonuç kümesini satır satır işlemek (her satıra özel iş kuralı uygulamak) gerektiğinde kullanılır.
- Satır bazlı güncellemede FOR UPDATE OF <kolon> ile tanımlanır ve güncelleme WHERE CURRENT OF ile yapılır.

Temel akış

DECLARE CURSOR … [FOR UPDATE OF <kolon>]

OPEN

Döngü: FETCH … INTO … → gerekli işlemler/güncellemeler

CLOSE

Kısa tanımlar

- **CURSOR:** Sorgu sonucunda geçerli satırı işaretleyen imleçtir.
- **OPEN:** Sorguyu başlatır; imleci ilk satırdan önce konumlandırır.
- **FETCH:** İmleci sonraki satıra taşır ve değerleri değişkenlere aktarır.
- **CLOSE:** İmleci kapatır, kaynakları serbest bırakır.

Örnek 2: Birden Çok Kayıt Getirme (retrieving multiple tuples)

```
0)  prompt("Enter the Department Name: " dname)
1)  EXEC SQL
2)  select DNUMBER into :dnumber
3)  from DEPARTMENT where DNAME = :dname ;
4)  EXEC SQL DECLARE EMP CURSOR FOR
5)  select SSN, FNAME, MINIT, LNAME, SALARY
6)  from EMPLOYEE where DNO = :dnumber
7)  FOR UPDATE OF SALARY ;
8)  EXEC SQL OPEN EMP ;
9)  EXEC SQL FETCH from EMP into :ssn, :fname, :minit, :lname, :salary
10) while (SQLCODE == 0) {
11) printf("Employee name is:", fname, minit, lname)
12) prompt("Enter the raise amount: raise)
13) EXEC SQL
14) update EMPLOYEE
15) set SALARY = SALARY + :raise
16) where CURRENT OF EMP ;
17) EXEC SQL FETCH from EMP into :ssn, :fname, :minit, :lname, :salary
18) }
19) EXEC SQL CLOSE EMP ;
```

/* 0) Girdi: Departman Adı */

/* 1-3) Departman numarasını bul */

/* 4-7) Cursor tanımı — salary güncelleneceği için FOR UPDATE */
4-7 (Compile Time)

/* 8) Cursor'ı Aç */

/* 9) İlk FETCH */

/* 10–18) Döngü:
SQLCODE==0 olduğu sürece devam */
(SQLCODE=0'sa fetchin sonucu TRUE. Yani sonuç dönmüştür.)
8–18 (Run Time)

/* 19) Kapat */

Dynamic SQL

Amaç

- Çalışma zamanında (run-time) yeni (önceden derlenmemiş) SQL ifadelerini oluşturup çalıştırmak.
- Bir program, çalışma zamanında klavyeden SQL ifadeleri kabul eder. Kullanıcı klavyeden/konsoldan SQL girer.
- Bir işaretle ve tıkla işlemi belirli bir SQL sorgusuna dönüşür

Neden zor?

- Derleme zamanında sözdizimi/şema bilinmez → denetimler sınırlı.
- Dönen kolon sayısı/türü bilinmeyebilir (özellikle SELECT).
- Bu yüzden DML (INSERT/UPDATE/DELETE) görece kolay; serbest biçimli SELECT daha karmaşıktır.

Önemli ilkeler

- Hazırlama (PREPARE) → Çalıştırma (EXECUTE) modeli: plan önbelleği ve tekrar kullanım sağlar.
- Parametreleştirme şarttır (SQL injection'a karşı).
- Tek seferlik ifadeler için EXECUTE IMMEDIATE kullanılabilir.

Dynamic SQL – Örnek

```
EXEC SQL BEGIN DECLARE SECTION;
varchar sqlupdatestring[256];
EXEC SQL END DECLARE SECTION;
…
prompt (“Enter update command:“, sqlupdatestring);
EXEC SQL PREPARE sqlcommand FROM :sqlupdatestring;
EXEC SQL EXECUTE sqlcommand;
```

Java'da Gömülü SQL: SQLJ

Tanım: SQLJ, Java içinde gömülü SQL yazmaya yarayan bir standarttır.

- **Statik bağlama yapar:** host değişkenleri derleme zamanında bağlanır; plan/optimizasyon ön hazırlığı sayesinde çalışma zamanı hızlı olabilir.
- JDBC bunun tersidir: tamamen dinamik; SQL metni run-time'da hazırlanır.

Nasıl çalışır? Kaynakta #sql { ... } blokları yazılır.

- SQLJ translator bu blokları JDBC çağrılarına çevirir.
- **Yürütme akışı:** SQLJ → (çeviri) → JDBC → Sürücü → DB.

Artılar

- Kod okunaklıdır; sabit SQL için bazı hataları derleme zamanında yakalar.
- Ön hazırlık sayesinde bazı platformlarda performans avantajı sağlayabilir.

Eksiler

- Ek derleme zinciri (translator/profil) ve araç/IDE desteği sınırlı.
- Taşınabilirlik/ekosistem açısından günümüzde JDBC/ORM daha yaygın

Java'da Gömülü SQL: SQLJ

```
import java.sql.*;          // JDBC
import sqlj.runtime.*;      // SQLJ runtime
import sqlj.runtime.ref.*;  // DefaultContext vb.

// (Sürücü yüklü varsayımıyla)
Connection jdbcCon =
    DriverManager.getConnection("jdbc:postgresql://localhost:5432/company", "app", "****");

// SQLJ çalışma bağlamı
DefaultContext ctx = new DefaultContext(jdbcCon);
DefaultContext.setDefaultContext(ctx);
```

Örnek 1: Tek bir tuple'ı alma (retrieving single tuple)

```
string dname, ssn , fname, fn, lname, ln, bdate, address
char minit, mi ;
double salary, sal ;
integer dna, dnumber ;
ssn = readEntry (" Enter a Socia1 Security Number : ")
try {
#sql {select FNAME, MINIT, LNAME, ADDRESS, SALARY
into :fname , :minit, :lname, :address, :salary
from EMPLOYEE where SSN = :ssn} ;
} catch (SQLException se) {
System.out.println("Social Security Number does not exist: " + ssn)
Return ;
}
System.out.println(fname + " " + minit + " " + lname + " " + address + " " +salary)
```

Örnek 2: Adlandırılmış yineleyici ile birden fazla tuple'ı alma

- SQLJ iki tür yineleyiciyi destekler:
  - **Adlandırılmış yineleyici:** bir sorgu sonucuyla ilişkilendirilmiştir.
  - **Konumsal yineleyici:** bir sorgu sonucundaki yalnızca öznitelik türlerini listeler.

- FETCH işlemi, bir sorgu sonucundaki bir sonraki tuple'ı alır:
- fetch iterator-variable'ı program-variable'a dönüştür.

Örnek 2: Adlandırılmış yineleyici ile birden fazla tuple'ı alma

```
dname = readEntry("Enter the Department Name: ")
try {
#sql{select DNUMBER into :dnumber
from DEPARTMENT where DNAME = :dname}
} catch CSQLException se) {
System.out.println("Department does not exist: " + dname)
Return ;
}
System.out.printline("Employee information for Department: " + dname) ;
#sql iterator Emp(String ssn, String fname, String minit, String lname ,double salary) ;
Emp e = null ;
#sql e = {select ssn, fname, minit, lname, salary
from EMPLOYEE where DNO :dnumber}
while (e.next()) {
System.out.printline(e.ssn + " " + e.fname + " " + e.minit + " " + e.lname + " " + e.salary)
} ;
e.close() ;
```

Örnek 3: Konumsal yineleyici ile birden fazla tuple'ı alma

```
dname = readEntry("Enter the Department Name: ")
try {
#sql{select DNUMBER into :dnumber
from DEPARTMENT where DNAME = :dname}
} catch (SQLException se) {
System.out.println("Department does not exist: " + dname)
return ; }
System.out.printline("Employee information for Department: " + dname)
#sql iterator Emppos(String, String, String, String, double)
Emppos e = null ;
#sql e ={select ssn, fname, minit, lname, salary
from EMPLOYEE where DNO = : dnumber} ;

#sql {fetch :e into :ssn, :fn, :mi, :ln, :sal}
while (e.next()) {
System.out.printline(ssn + " " + fn + " " + mi + " " + ln + " " + sal)
#sql {fetch :e into :ssn, :fn, :mi, :ln, :sal}
};
e.close() ;
```

2) Fonksiyon Çağrılarıyla Veritabanı Programlama (API Yaklaşımı)

- **Embedded SQL:** statik (preprocessor gerekir).
- **API (JDBC/ODBC/SQL-CLI/ADO.NET/PDO/psycopg2):** dinamik veritabanı programlama; kütüphane fonksiyonları üzerinden SQL çalıştırılır.

Avantajlar

- Ön-işlemci yok → daha esnek ve taşınabilir.
- Aynı uygulama birden çok DBMS'e bağlanabilir (sürücü değiştir).
- Bağlantı havuzu, parametrik sorgular, transaction yönetimi, hata yakalama ekosistemi güçlü.

Dezavantajlar

- Sözdizimi/tip kontrolleri çalışma zamanında ortaya çıkar.
- SELECT sonuçlarının kolon sayısı/türü önceden bilinmeyebilir → ek kod (metadata/descriptor) gerekir.

2) Fonksiyon Çağrılarıyla Veritabanı Programlama (API Yaklaşımı)

- **API yaklaşımı** = "fonksiyon çağrısı → sürücü → DBMS". Kaynak kodda precompile yok.
- **Güvenlik:** DİNAMİK SQL'de string birleştirme yerine parametre (injection'a karşı).
- **Taşınabilirlik:** Sürücü değişimiyle Postgres/Oracle/SQL Server arasında geçiş mümkün (SQL diyalekti farklarına dikkat).
- **Performans:** Pool, batch/executeBatch, server-side cursor/streaming okuma, autocommit=off + kısa transaction.

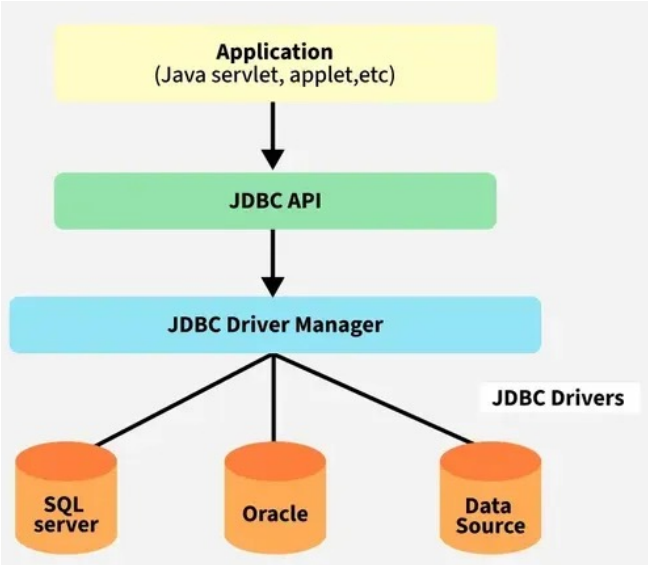
2) Fonksiyon Çağrılarıyla Veritabanı Programlama (API Yaklaşımı)

SQL/CLI (Call-Level Interface)

- SQL standardının bir parçasıdır. ODBC (Open Database Connectivity) tarzı fonksiyon çağrılarıyla SQL
- Aynı program içinde birden fazla veritabanına kolay erişim sağlar.
- Belirli kütüphanelerin (örneğin, C için sqlcli.h) kurulu ve kullanılabilir olması gerekir.
- SQL ifadeleri dinamik olarak oluşturulur ve çağrılarda dize parametreleri olarak geçirilir.

JDBC

- Java programlama için SQL bağlantı fonksiyonu çağrılarıdır.
- JDBC fonksiyonlarına sahip bir Java programı, JDBC sürücüsü olan herhangi bir ilişkisel DBMS'ye erişebilir.
  - JDBC Sürücü: JDBC API'sinin belirli bir uygulama işlevi)
- JDBC, bir programın birden fazla veritabanına (veri kaynağı olarak bilinir) bağlanmasına olanak tanır.



JDBC: Connection → PreparedStatement → ResultSet
SQL/CLI: Connection → Statement → Fetch.

JDBC Sınıfları: Akış

Driver → Connection → Statement/PreparedStatement/CallableStatement → ResultSet

1) Driver

- JDBC kütüphanesi java.sql.* ile gelir; sürücü (PostgreSQL, Oracle, SQL Server vb.) JAR olarak eklenir.
- Modern JDBC'de çoğu sürücü Service Provider ile otomatik yüklenir (çoğu zaman Class.forName(...) gerekmez).

2) Connection

- DriverManager.getConnection(...) veya pool (HikariCP vb.) ile alınır.
- Ayarlar: setAutoCommit(false), setTransactionIsolation(...), setReadOnly(true) gibi.
- Bağlantı pahalıdır → üretimde connection pooling şart.

JDBC Sınıfları: Akış

Driver → Connection → Statement/PreparedStatement/CallableStatement → ResultSet

3) Statement Ailesi

- Statement: ham SQL (parametre yok). Güvenlik için önerilmez.
- PreparedStatement: (soru işaretleriyle belirtilir) ? parametreleriyle güvenli ve hızlı (plan yeniden kullanılır). Standart tercih.
- CallableStatement: stored procedure çağrıları, IN/OUT parametreleri.

4) ResultSet

- Sorgu sonuçları satır/sütun tablosu olarak döner.
- Türler: TYPE_FORWARD_ONLY (varsayılan), TYPE_SCROLL_INSENSITIVE/ SENSITIVE; eşzamanlılık: CONCUR_READ_ONLY/UPDATABLE.
- Önemli ayarlar: setFetchSize(n) (akışlı okuma), getGeneratedKeys() (oto-ID).

Örnek 1: Tek bir tuple'ı alma (retrieving single tuple)

```
import java.io.*
import java.sql.*
class getEmplnfo {
public static void main (String args []) throws SQLException, IOException {
try { Class.forName("oracle.jdbc.driver.OracleDriver")
} catch (ClassNotFoundException x) {
System.out.println ("Driver could not be loaded") ;}
String dbacct, passwrd, ssn, lname;
Double salary ;
dbacct = readentry("Enter database account:");
passwrd = readentry("Enter pasword:") ;
Connection conn = DriverManager.getConnection) ("jdbc:oracle:oci8:" + dbacct + "/" + passwrd)
String stmt1 = "select LNAME, SALARY from EMPLOYEE where SSN = ?"
PreparedStatement p = conn.prepareStatement(stmt1) ;
ssn = readentry("Enter a Social Security Number: ") ;
p.clearParameters() ;
p.setString(1, ssn) ; //bounding
ResultSet r = p.executeQuery()
while (r.next()) {
lname = r.getString(l) ;
salary = r.getDouble(2) ;
system.out.printline(lname + salary);} } }
```

Örnek 2: Birden Çok Kayıt Getirme (retrieving multiple tuples)

```
import java. io.* ;
import java. sql .*,
class printDepartmentEmps {
public static void main (String args [J) throws SQLException, IOException {
try { Class. forName("oracle. jdbc ,driver.OracleDriver")
} catch (ClassNotFoundException x) { ,
System.out.println ("Driver could not be loaded“);}
String dbacct, passwrd, name ;
Double salary;
Integer dno ;
dbacct = readentry("Enter database account: ")
passwrd = readentry("Enter pasword: ") ;
Connection conn = DriverManager.getConnection ("jdbc:oracle:oci8:" + dbacct + "/" + passwrd)
dno = readentry("Enter a Department Number: ") ;
String q = "select LNAME, SALARY from EMPLOYEE where DNO =“+dno.tostring() ;
Statement s = conn. createStatement();
ResultSet r = s. executeQuery(q)
while (r.next()) {
name = r. getStri ng(l) ;
salary = r.getDouble(2) ;
system.out.printline(lname + salary) }}}
```

YILDIZ TEKNİK ÜNİVERSİTESİ

Yasemin Topuz

Yıldız Teknik Üniversitesi

ytopuz@yildiz.edu.tr