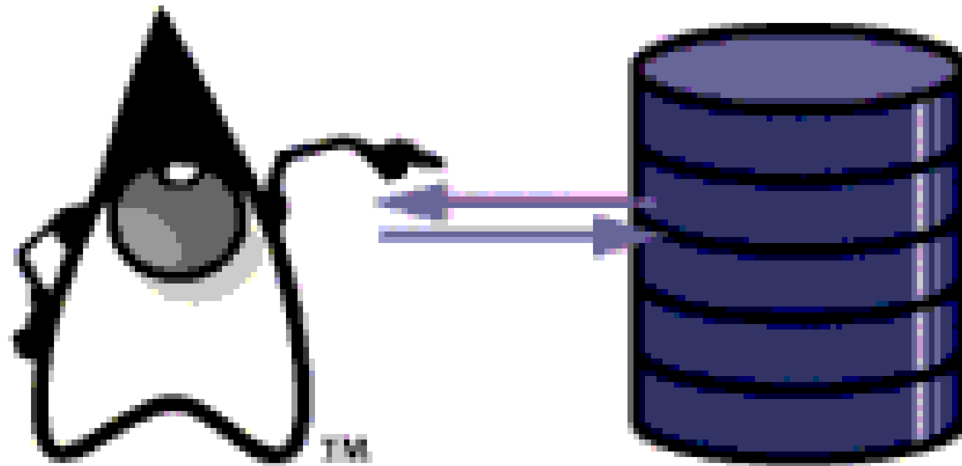
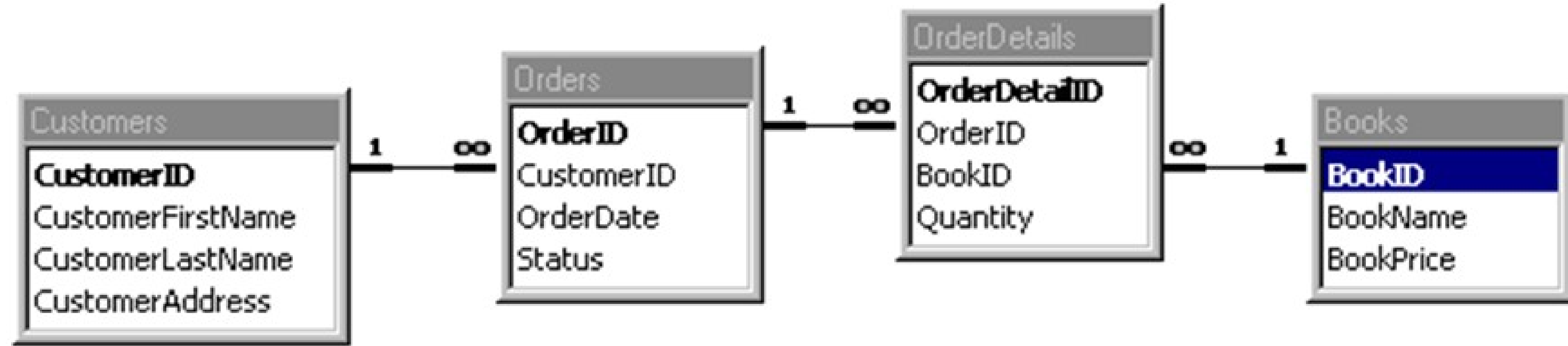
Delo s podatkovnimi bazami
(JDBC)



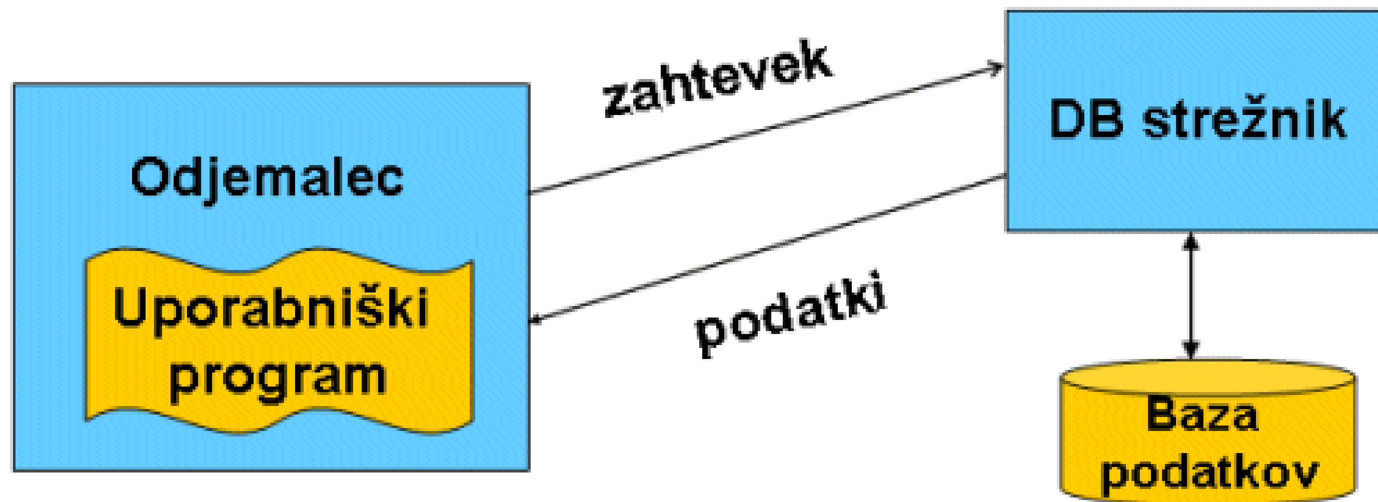
Primer preproste podatkovne baze



Nekaj pojasnil:
Stranka (Customer) ima lahko več naročil (Orders).
Posamezno naročilo (Order) ima lahko več zapisov OrderDetail.
Vsak zapis OrderDetail se nanaša na natančno eno knjigo (Book).

Vsi primarni ključi (polja ID) so avtomatsko generirani in jih zato pri vrivanju zapisov ne navajamo.
OrderDetail.Quantity je tipa int.
Books.BookPrice je tipa double .
CustomerFirstName, CustomerLastName CustomerAddress in BookName are so tipa String.
Orders.OrderDate je objekt tipa Date/Time

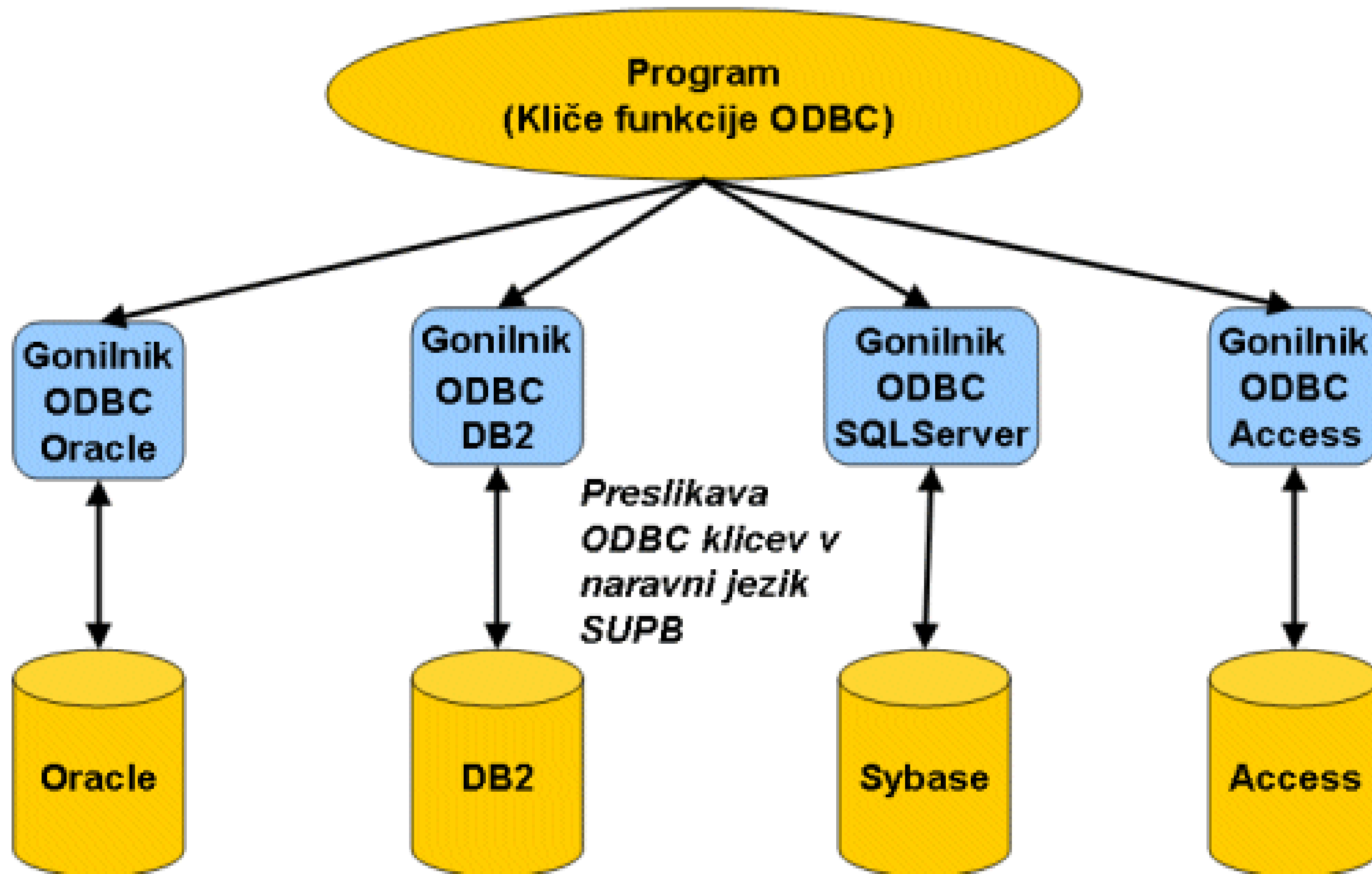
Interakcija s podatkovno bazo



Obravnavali bomo, kako lahko nek uporabniški program interaktira s podatkovno bazo.

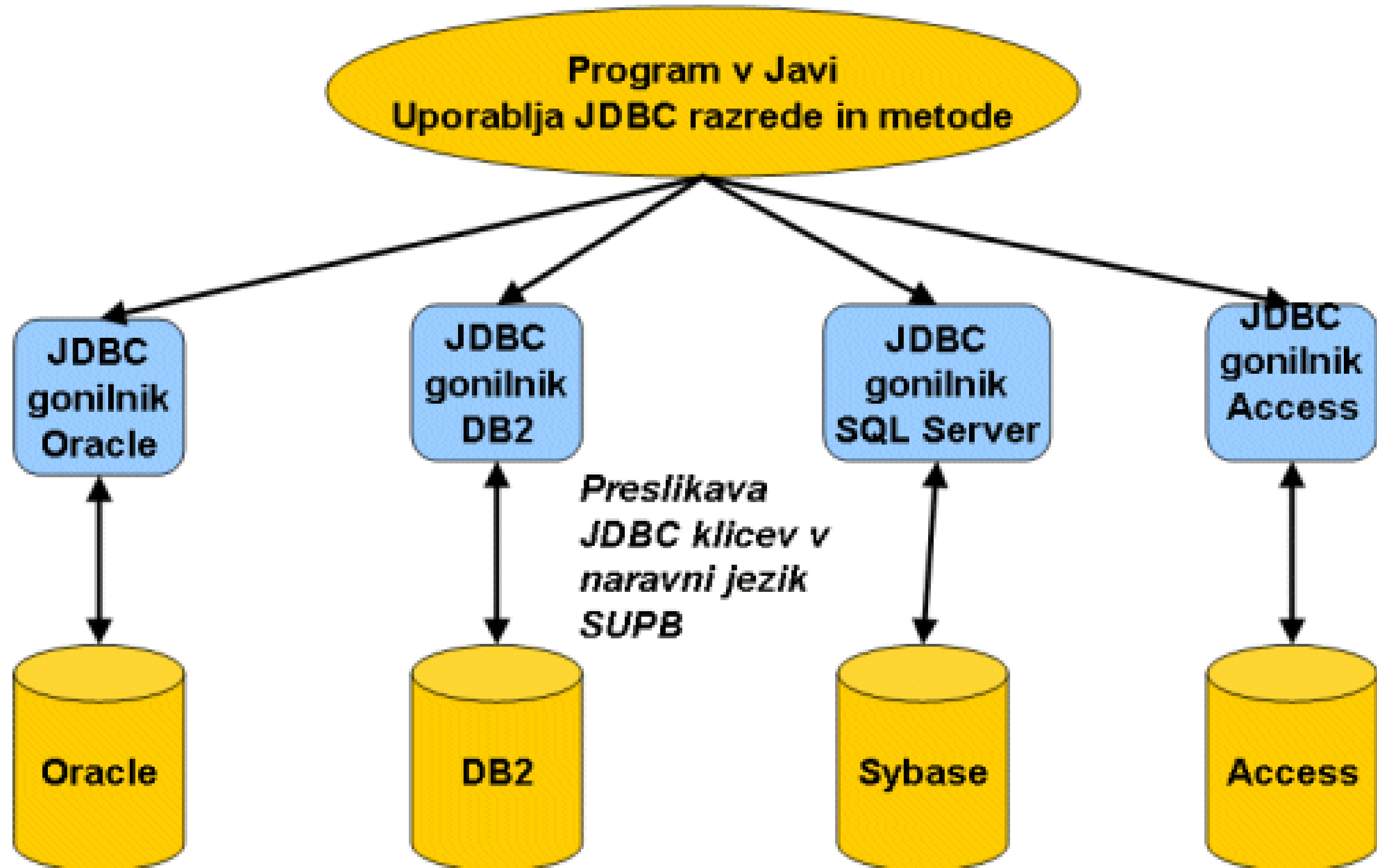
Programi so seveda pisani v različnih programskih jezikih in tečejo v različnih okoljih. Prav tako poznamo različne sisteme za upravljanje podatkovnih baz.

Standard ODBC



Microsoft je zato predlagal standard ODBC **(O**pen **D**ata **B**ase **C**onnectivity), ki je namenjen komunikaciji uporabniških programov s podatkovnimi bazami.

Programi v Javi in JDBC

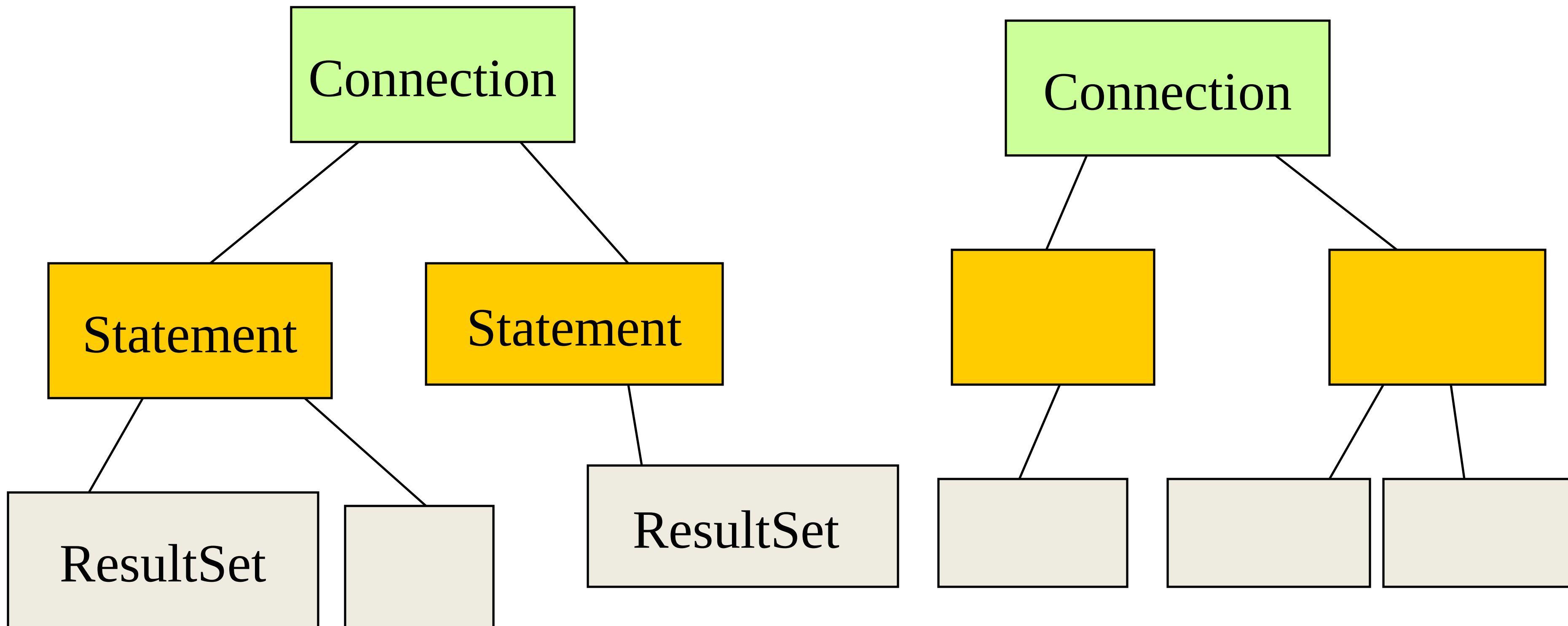


Programi v Javi in dostop do podatkovnih baz s pomočjo JDBC

6 korakov za uporabo JDBC

1. Naložimo JDBC Driver
2. Vzpostavimo povezavo s podatkovno bazo (Database Connection)
3. Tvorimo objekt Statement
4. Izvedemo povpraševanje (Query)
5. Obdelamo rezultat (Result)
6. Zapremo povezavo (Connection)

Terminologija (pomen objektov)



- Program ima lahko eno ali več povezav (connections) do strežnikov podatkovnih baz
- Eni povezavi (connection) je lahko prirejen eden ali več stavkov (statement)
- Enemu stavku (statement) je lahko prirejenih eden ali več rezultatov (result).

Dostop do podatkovne baze preko JDBC

1. Tvoriti moramo objekt ***Connection*** , ki je povezan s podatkovno bazo
2. Tvorimo objekt ***Statement*** , s katerim odpremo objekt *Connection*
3. Povpraševalno izvajanje objekta *Statement* vrne objekt ***ResultSet***
4. Sedaj lahko obdelamo vrstice v objektu *ResultSet*

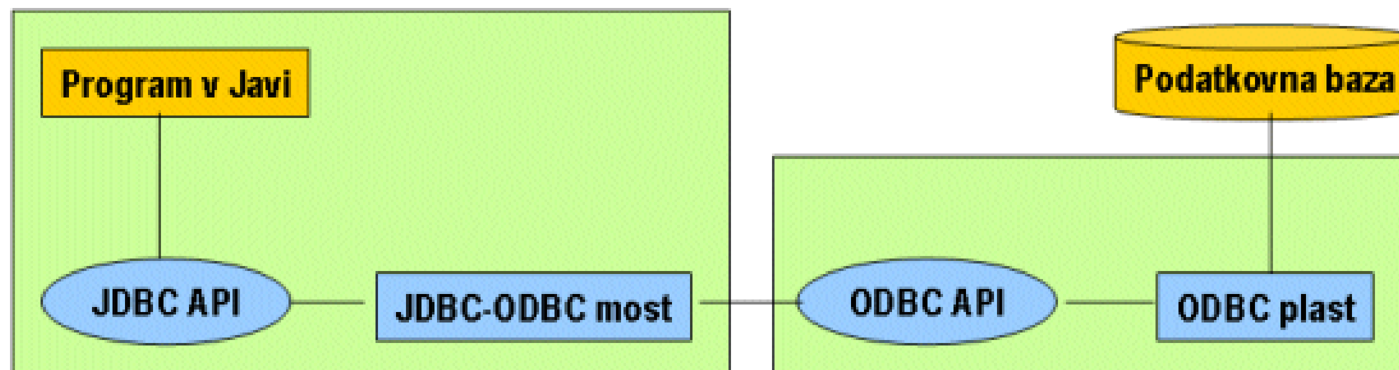
Glavni paket, ki ga moramo uporabiti, je *java.sql*. Zato na začetku programa uporabimo stavek

```
import java.sql.* ;
```

Ta paket spada v standardno distribucijo JDK (Java Development Kit). Obstaja pa tudi njegova bolj napredna razširitev *javax.sql*, ki pa jo redko uporabljamo.

Gonilniki (drivers)

Posamezni sistemi za upravljanje s podatkovnimi bazami zahtevajo za svoj dostop z vmesnikom JDBC ustrezen JDBC gonilnik.

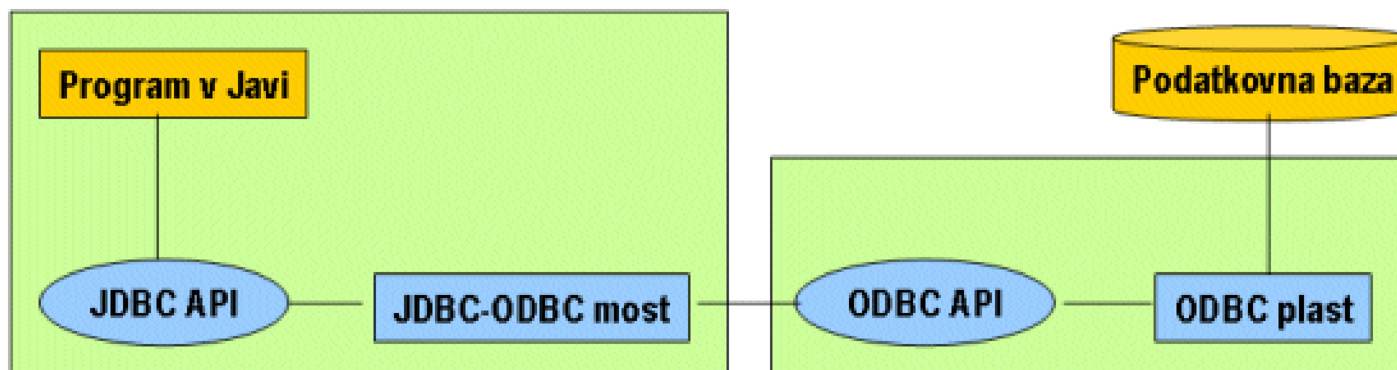


Sun ponuja kot del svoje standardne distribucije premostitveni gonilnik JDBC-ODBC.

Ta nam omogoča povezavo s katerokoli podatkovno bazo, ki razume ODBC.

Nalaganje gonilnika

Če želimo dostopiti do podatkovne baze, moramo najprej naložiti gonilnik. Seveda mora biti gonilnik nameščen na sistem (kar pomeni, da mora biti nameščena ustrezna »jar« datoteka z gonilnikom in navedena v poti (classpath)).



Če uporabljamo premostitveni gonilnik JDBC-ODBC, moramo še našo podatkovno bazo registrirati z ODBC

Dinamično nalaganje gonilnika

Dinamično nalaganje poteka med samim izvajanjem programa:

```
Properties props = new Properties() ;
FileInputStream in = new FileInputStream("Database.Properties") ;
props.load(in) ;
String drivers = props.getProperty("jdbc.drivers") ;
Class.forName(drivers) ;
```

Izgled datoteke Database.properties

```
# Default JDBC driver and database specification
jdbc.drivers      =  sun.jdbc.odbc.JdbcOdbcDriver
database.Shop  =  jdbc:odbc:Shop
```

Tvorba povezave (Connection)

Povezavo dobimo tako, da navedemo URL konkretne podatkovne baze, ki jo želimo uporabiti. Oblika tega URL je odvisna od uporabljenega gonilnika. URL podatkovne baze lahko dobimo potem, ko je gonilnik naložen in lahko uporabimo prej omenjeno datoteko z lastnostmi (properties).

```
String database = props.getProperty("database.Shop") ;
Connection con = DriverManager.getConnection(database) ;
```

Če uporabimo premostitveni gonilnik JDBC-ODBC, je URL podatkovne baze jdb:odbc:xxx , pri čemer je xxx podatkovni vir ODBC, registriran za našo podatkovno bazo. (ime lastnosti, ki jo uporabimo, ni pomembno).

Uporaba objektov Statement

Objekt Statement (izjava) dobimo iz povezave z naslednjim stavkom:

```
Statement stmt = con.createStatement() ;
```

Z objektom Statement lahko nato izvajamo ali nadziramo izvajanje različnih povpraševanj SQL.

- Uporabimo ***stmt.executeUpdate*** z argumentom v obliki niza, ki vsebuje SQL stavek za posodobitev (INSERT, DELETE or UPDATE). Stavek vrne število posodobljenih vrstic.
- Uporabimo ***stmt.executeQuery*** z argumentom v obliki niza, ki vsebuje SQL povpraševanje SELECT. Stavek vrne objekt ***ResultSet***, ki ga nato uporabimo za dostop do vrstic rezultata povpraševanja.
- Uporabimo ***stmt.execute*** za izvajanje poljubnega SQL stavka kateregakoli tipa. Je pa pri tem stavku težje ugotavljanje rezultata, bodisi celega števila bodisi objekta *ResultSet*. Ta stavek uporabimo, če želimo posplošen dostop do podatkovne baze oziroma za programirano tvorbo povpraševanj.

Uporaba objekta statement

```
int count = stmt.executeUpdate("INSERT INTO Customers "  +
                    "(CustomerFirstName, CustomerLastName, CustomerAddress) "
                    "VALUES ('Tony', 'Blair', '10 Downing Street, London')") ;
ResultSet rs = stmt.executeQuery("SELECT * FROM Customers") ;// do something with count and RS
```

Sintaksa stavkov SQL, ki jih posredujemo kot argument v obliki niza, se mora ujemati s sintakso, ki jo uporablja navezana podatkovna baza.

Primer:

```
ResultSet rs = stmt.executeQuery("SELECT * FROM Customers" +
                    "WHERE CustomerLastName = 'O''Neill'") ;
```

Uporaba objektov ResultSet (1)

*Če ne poznamo točne strukture tabel (sheme) v ResultSet, jo lahko dobimo preko objekta **ResultSetMetaData**.*

```
ResultSetMetaData rsmd = rs.getMetaData() ;
int colCount = rsmd.getColumnCount() ;
for (int i = 1 ; i <= colCount ; i++) {
    if (i > 1) out.print(", ");
    out.print(rsmd.getColumnLabel(i)) ;
}out.println() ;
```

Ko dobimo objekt ResultSet , lahko iz njega dobimo njegove vrstice oziroma polja v njegovih vrsticah:

```
while (rs.next()) {
    for (int i = 1 ; i <= colCount ; i++) {
        if (i > 1) out.print(", ");
        out.print(rs.getObject(i)) ;
    }
    out.println() ;
}
```

Uporaba objektov ResultSet (2)

Stolpce začnemo šteti od 1 dalje in ne od 0, kot je sicer navada pri poljih v Javi. Lahko pa uporabimo metodo getObject na objektu ResultSet . Ta metoda sprejme kot argument ime stolpca. Na voljo imamo tudi metode tipa getxxx , v katerih namesto številke stolpca navedemo kot argument ime stolpca. Zato lahko zgornjo kodo zapišemo tudi tako:

```
while (rs.next()) {
        out.println(   rs.getObject("CustomerID")       + ", " +
                       rs.getObject("CustomerFirstName") + ", " +
                       rs.getObject("CustomerLastName")  + ", " +
                       rs.getObject("CustomerAddress") ) ;
}
```

Objekti PreparedStatement

Te objekte lahko uporabimo namesto objektov Statement. Objekti PreparedStatement imajo naslednje prednosti:

1.Pri ponavljanih povpraševanjih (na primer v zankah) je to bolj učinkovito, ker so stavki SQL prevedeni le enkrat

2.Mehanizem vključevanja vrednosti parametrov poskrbi za vse posebne znake

```
PreparedStatement pstmt = con.prepareStatement(
                    "INSERT INTO Customers " +
                    "(CustomerFirstName, CustomerLastName, CustomerAddress) "+
                    "VALUES (?, ?, ?)") ;
pstmt.clearParameters() ;
pstmt.setString(1, "Joan") ;
pstmt.setString(2, "D'Arc") ;
pstmt.setString(3, "Tower of London") ;
count = pstmt.executeUpdate() ;
System.out.println ("\nInserted " + count + " record successfully\n") ;

pstmt.clearParameters() ;
pstmt.setString(1, "John") ;
pstmt.setString(2, "D'Orc") ;
pstmt.setString(3, "Houses of Parliament, London") ;
count = pstmt.executeUpdate() ;
System.out.println ("\nInserted " + count + " record successfully\n") ;
```

Transakcije so mehanizem za skupinske operacije v tem smislu, da uspejo vse ali pa nobena.

To prepreči nekonsistenčne probleme v podatkovni bazi, ki bi nastali, če bi se uspešno izvedel le del operacij.

Pomislimo na primer, da bi želeli prestaviti del denarja iz enega računa na drugi in da bi dvig denarja uspel, polog na drugi račun pa ne. Žalostno, kaj ne? Če to izvedemo kot transakcijo, je prva operacija (dvig) veljavna le, če tudi drugi del (polog) uspe.

Transakcije in JDBC

Ko smo dobili povezavo »*Connection« je lastnost AutoCommit* nastavljena na *true*. To pomeni, da je vsaka izvedba povpraševanja zapomnjena takoj po svoji izvedbi in preden začnemo izvajati naslednje povpraševanje. Če želimo združevati operacije v transakcije, moramo lastnost *AutoCommit* izključiti:

```
con.setAutoCommit(false) ;
```

Sedaj moramo dobiti nov objekt Statement (stari ne bo več deloval) in ga uporabljamo na že znani način. Ko zaključimo v se želene operacije, zahtevamo, njihovo pomnenje z naslednjim ukazom:

```
con.commit() ;
```

Če pride med izvajanjem transakcije do težav (recimo, da nismo mogli izvesti vseh operacij transakcije), lahko vse dotedanje operacije razveljavimo z ukazom:

```
con.rollBack() ;
```

Popoln primer: podatkovna baza o kavici

1. Vzpostavitev tabele (preko JDBC)
2. Vključevanje zapisov (preko JDBC)
3. Poizvedovanje (preko JDBC)

Vzpostavitev tabele (1)

```
import java.sql.*;

public class CreateCoffees {
    public static void main(String args[]) {
        String url = "jdbc:mysql://localhost/cerami";
        Connection con;
        String createString;
        createString = "create table COFFEES " +
                        "(COF_NAME VARCHAR(32),
" +
                        "SUP_ID INTEGER, " +
                        "PRICE FLOAT, " +
                        "SALES INTEGER, " +
                        "TOTAL INTEGER)";
        Statement stmt;
```

Vzpostavitev tabele (2)

```
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch(java.lang.ClassNotFoundException e) {
            System.err.print("ClassNotFoundException: ");
            System.err.println(e.getMessage());
        }

        try {
            con = DriverManager.getConnection(url);
            stmt = con.createStatement();
            stmt.executeUpdate(createString);
            stmt.close();
            con.close();

        } catch(SQLException ex) {
            System.err.println("SQLException: " + ex.getMessage());
        }
    }
}
```

Vključevanje zapisov (1)

```
import java.sql.*;

public class InsertCoffees {

        public static void main(String args[]) throws SQLException
{
                System.out.println ("Adding Coffee Data");
                ResultSet rs = null;
                PreparedStatement ps = null;
                String url = "jdbc:mysql://localhost/cerami";
                Connection con;
                Statement stmt;
                try {
                        Class.forName("org.gjt.mm.mysql.Driver");
                } catch(java.lang.ClassNotFoundException e) {
                        System.err.print("ClassNotFoundException: ");
                        System.err.println(e.getMessage());
                }
```

Vključevanje zapisov (2)

```
try {
        con = DriverManager.getConnection(url);
        stmt = con.createStatement();
        stmt.executeUpdate ("INSERT INTO COFFEES " +
            "VALUES('Amaretto', 49, 9.99, 0, 0)");
        stmt.executeUpdate ("INSERT INTO COFFEES " +
            "VALUES('Hazelnut', 49, 9.99, 0, 0)");
        stmt.executeUpdate ("INSERT INTO COFFEES " +
            "VALUES('Amaretto_decaf', 49, 10.99, 0, 0)");
        stmt.executeUpdate ("INSERT INTO COFFEES " +
            "VALUES('Hazelnut_decaf', 49, 10.99, 0, 0)");
        stmt.close();
        con.close();
        System.out.println ("Done");
    } catch(SQLException ex) {
        System.err.println("-----SQLException-----");
        System.err.println("SQLState:  " + ex.getSQLState());
        System.err.println("Message:  " + ex.getMessage());
        System.err.println("Vendor:  " + ex.getErrorCode());
    }
  }}
```

Poizvedovanje (1)

```
import java.sql.*;

public class SelectCoffees {

    public static void main(String args[]) throws SQLException {
        ResultSet rs = null;
        PreparedStatement ps = null;
        String url = "jdbc:mysql://localhost/cerami";
        Connection con;
        Statement stmt;
        try {
            Class.forName("org.gjt.mm.mysql.Driver");
        } catch(java.lang.ClassNotFoundException e) {
                System.err.print("ClassNotFoundException: ");
                System.err.println(e.getMessage());
        }

        try {
            con = DriverManager.getConnection(url);
                stmt = con.createStatement();
```

Poizvedovanje (2)

```
        ResultSet uprs = stmt.executeQuery("SELECT * FROM COFFEES")
          System.out.println("Table COFFEES:");
          while (uprs.next()) {
                String name = uprs.getString("COF_NAME");
                int id = uprs.getInt("SUP_ID");
                float price = uprs.getFloat("PRICE");
                int sales = uprs.getInt("SALES");
                int total = uprs.getInt("TOTAL");
                System.out.print(name + "   " + id + "   " + price);
                System.out.println("   " + sales + "   " + total);
          }
          uprs.close();
          stmt.close();
          con.close();

    } catch(SQLException ex) {
          System.err.println("-----SQLException-----");
          System.err.println("SQLState:  " + ex.getSQLState());
          System.err.println("Message:  " + ex.getMessage());
          System.err.println("Vendor:  " + ex.getErrorCode());
} }}
```