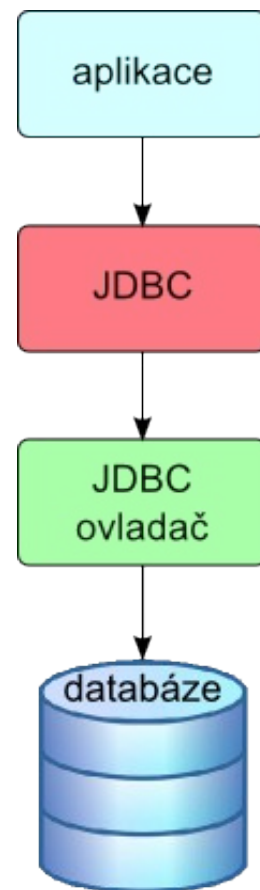




JDBC (Java Database Connectivity)

- **JDBC (Java DataBase Connectivity)** je rozhraní pro komunikaci Java aplikací s různými databázovými servery.
 - S různými databázovými servery se pracuje stejným způsobem, jenom SQL dotazy a příkazy se liší.
- Abyste se mohli připojit přes JDBC k nějaké databázi, tak si musíte pro příslušný databázový server stáhnout JDBC ovladač :
 - Zadejte do Google: jdbc <název databáze>
 - Obvykle stáhnete JAR soubor s ovladačem nebo ZIP archiv, ve kterém se zmíněný JAR soubor nachází.





Práce s databází

- Práce s databází se skládá z pěti kroků:
 - Přidání JAR souboru, ve kterém se nachází JDBC ovladač pro konkrétní databázi do Build Path projektu.
 - Získání objektu typu `Connection` (navázání připojení do databáze).
 - Získání příkazu (`Statement` / `PreparedStatement`) z `connection`.
 - Spuštění SQL příkazu / dotazu.
 - Uzavření příkazu.
- Poznámka:
 - Navázání připojení do databáze je časově náročná operace (minimálně v řádu stovek milisekund), je tudíž vhodné si toto připojení někde uložit a používat ho celou dobu běhu aplikace.



Získání připojení do databáze I.

- Každý databázový výrobce vyžaduje malinko něco jiného, tudíž je v případě potíží vždy vhodné konzultovat dokumentaci k příslušnému JDBC ovladači (obvyčejně ji stáhnete rovnou s JDBC ovladačem).
- Prakticky ale existují dva způsoby jak získat objekt typu Connection.

- Buď pomocí třídy DriverManager, což je starší způsob:


```
String dbUrl = "jdbc:hsqldb:hsql://localhost/eshop";  
String user = "sa";  
String password = "";  
Connection connection = DriverManager.getConnection  
                                (dbUrl, user, password);
```

← JDBC URL, specifické
pro každého DB výrobce



Získání připojení do databáze II.

- Nebo pomocí objektu typu `DataSource`. Níže je příklad připojení do databáze Apache Derby:



```
JDBCDataSource dataSource = new JDBCDataSource();  
dataSource.setUrl("jdbc:hsqldb:hsqldb://localhost/eshop");  
dataSource.setUser("sa");  
dataSource.setPassword("");  
Connection connection = dataSource.getConnection();
```

Třída, která
se nachází
v JDBC
ovladači
HSQL
databáze

- Celá řada databázových výrobců přibaluje třídy implementující `DataSource` do JAR balíčku obsahující JDBC ovladač.



Získání připojení do databáze III.

- Také existují open source DataSource knihovny nezávislé na databázi (například Apache DBCP nebo C3P0).
- Spousta DataSource tříd podporuje tzv. databázový pooling. Při tom není vytvořené pouze jedno připojení do databáze, ale více. Používá se to zejména u webových aplikací, kde by se pouze jedno připojení do databáze stalo při přístupu mnoha klientů úzkým hrdlem celé aplikace.
- Nezapomínejte uzavírat příkazy (Statement / PreparedStatement)!



Připojení k DB pomocí DataSource I.

- Příklad použití `oracle.jdbc.pool.OracleDataSource`:

```
OracleDataSource dataSource = new OracleDataSource();
dataSource.setURL("jdbc:oracle:thin:@localhost:1521:XE");
dataSource.setUser("user");
dataSource.setPassword("password");
// nastavení connection pooling:
dataSource.setConnectionCachingEnabled(true);
dataSource.setConnectionCacheName("MYCACHE");
Properties cacheProps = new Properties();
cacheProps.setProperty("MinLimit", "1");
cacheProps.setProperty("MaxLimit", "4");
cacheProps.setProperty("InitialLimit", "1");
dataSource.setConnectionCacheProperties(cacheProps);
Connection connection = dataSource.getConnection();
```



Připojení k DB pomocí DataSource II.

- Příklad použití `org.apache.commons.dbcp.BasicDataSource`:

```
BasicDataSource dataSource = new BasicDataSource();
dataSource.setDriverClassName("oracle.jdbc.driver.OracleDriver");
dataSource.setUrl("jdbc:oracle:thin:@localhost:1521:XE");
dataSource.setUsername("user");
dataSource.setPassword("password");
// nastavení connection pooling:
dataSource.setMaxActive(100);
dataSource.setMaxIdle(30);
dataSource.setInitialSize(5);
Connection connection = dataSource.getConnection();
```


Vykonání SQL příkazu I.

- Pomocí třídy Statement:

Tyto data typicky
získáme od uživatele

```
String name = "Java For Beginners";  
String description = "Training";  
int price = 22000;  
String sql = "insert into item (name, description, " +  
             "price) values ( '" + name +  
             "', '" + description + "', " +  
             price + ")";  
  
Statement statement = connection.createStatement();  
statement.execute(sql);  
statement.close();
```

Vykonání SQL příkazu II.

- Pomocí třídy PreparedStatement:

```
String sql =
```

```
    "insert into item (name, description, price) values (?, ?, ?)";
```

```
PreparedStatement preparedStatement =
```

```
    connection.prepareStatement(sql);
```

```
preparedStatement.setString(1, "Java For Beginners");
```

```
preparedStatement.setString(2, "Training");
```

```
preparedStatement.setInt(3, 22000);
```

Místo reálných dat
dáme otazníky



Reálná data
poté dosadíme
později

Pořadí otazníku v SQL příkazu,
číslováno od jedničky.

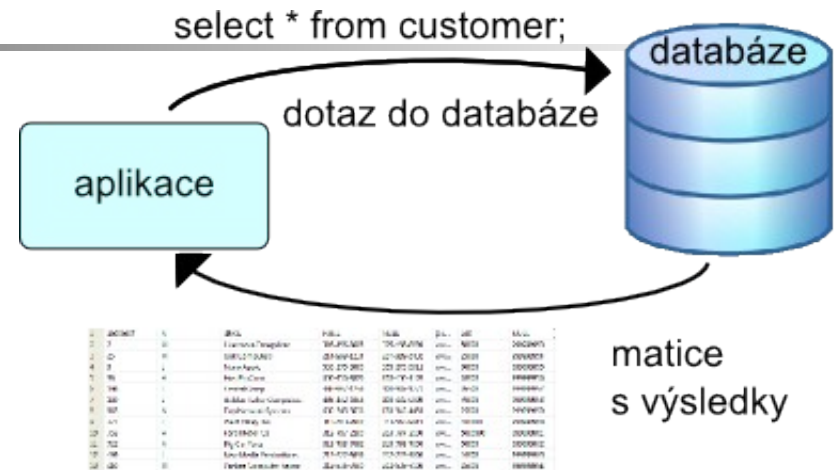


Statement vs. PreparedStatement

- Používejte pokud možno vždy třídu PreparedStatement.
 - **Je přehlednější** – vše se „nemastí“ do jednoho Stringu.
 - **Je rychlejší** – databázové servery jsou schopné šablonu příkazu nacachovat a jenom mění dynamická data uvnitř.
 - **Je bezpečnější** – Statement není imunní vůči útoku typu SQL Injection, zatímco PreparedStatement vůči tomuto typu útoku imunní je.

Vykonání SQL dotazu

K získání dat z databáze se používá metoda `executeQuery()`, která vrací objekt typu `ResultSet`:



Získáme matici
s výsledky z DB

```
String sql = "select * from item";
```

```
PreparedStatement statement =  
    connection.prepareStatement(sql);
```

```
ResultSet resultSet = statement.executeQuery();
```

```
while (resultSet.next()) {  
    System.out.println(resultSet.getString("name"));  
}  
statement.close();
```

Voláme metodu
`executeQuery()`

V cyklu se
procházejí
řádky matice

Získáváme hodnoty ze sloupců matice



Zapnutí transakcí

- Po vytvoření připojení k databázi (získání objektu typu `Connection`) je nutné nastavit příznak určující, že ukončování transakcí budeme řídit sami (jinak by se prováděl `commit` za každým jednotlivým SQL dotazem):

```
connection.setAutoCommit(false);
```

- Transakce se potvrdí, resp. zruší následujícími příkazy:

```
connection.commit();  
connection.rollback();
```