

**Спецификация JDBC. Достъп и  
работа с база от данни. Добри  
практики.**

# Съдържание

- Въведение в JDBC
- JDBC Заявки
- Работа с транзакции в JDBC
- Добри практики

# **JDBC**

## **Въведение**

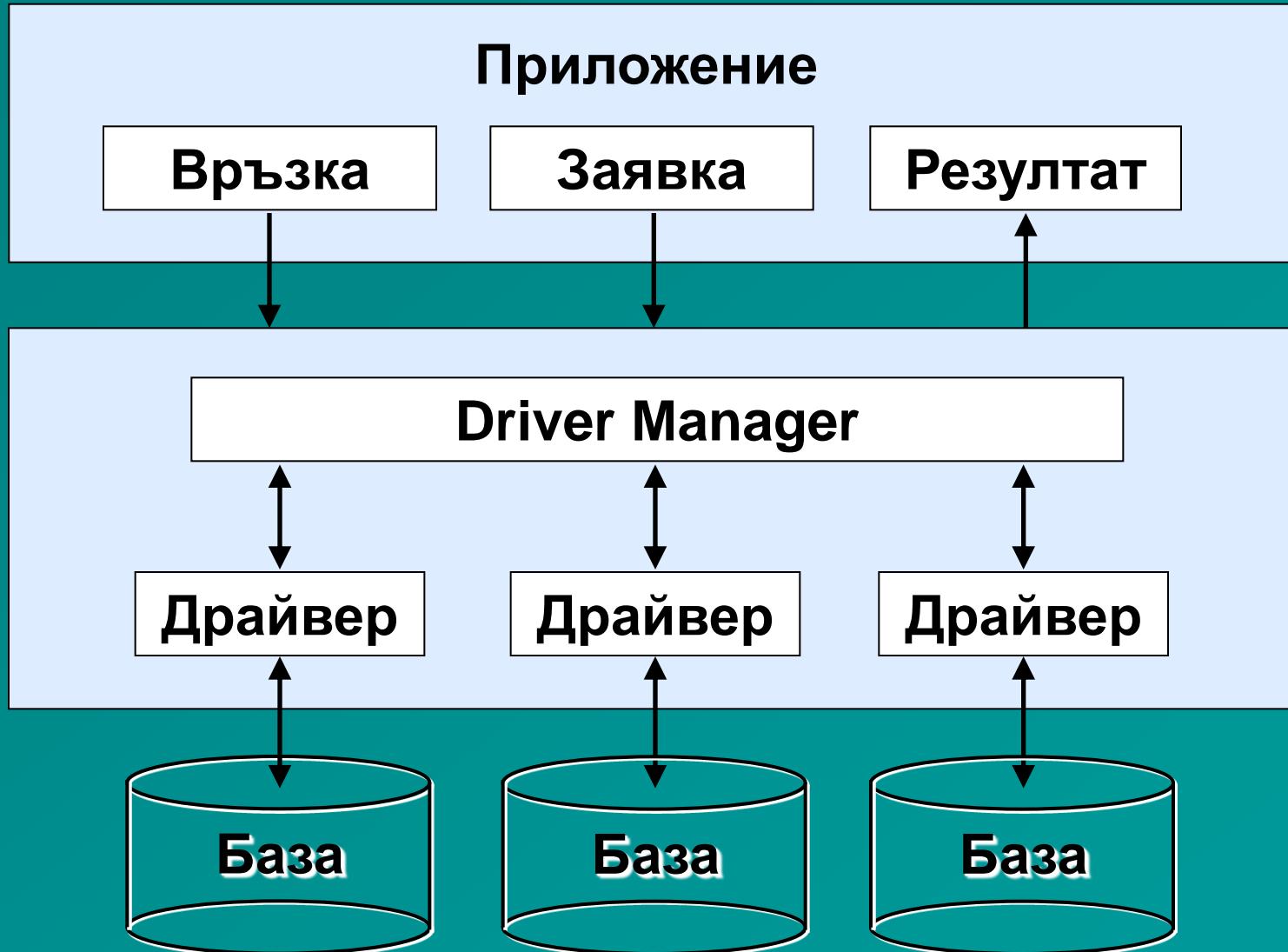
# JDBC

- Интерфейс за достъп и работа с релационни бази данни през Java.
- Базиран на X/Open SQL CLI.

# **Версии**

- JDBC Core API в  
`java.sql`
- JDBC 2.1 и 2.0 Optional Package  
`javax.sql`
- JDBC 3.0 API  
включва горните два
- JDBC 4.0

# Архитектурата на JDBC



# Какво прави JDBC?

- Установяване на връзка към БД
- Изпълнение на SQL заявка
- Обработка на резултатите

# Установяване на връзка към БД

- Зареждане на драйвер
- Установяване на връзка

# Зареждане на драйвер

- Клас DriverManager
- Интерфейс DataSource
  - ConnectionPoolDataSource
  - XADatasource

# Зареждане на драйвер

- Зареждане на JDBC драйвер с един ред:

```
Class.forName("<jdbc driver class>");
```

- Името на нужния ви клас, ще намерите в документацията на драйвера

# Установяване на връзка

- Връзка (Connection)
  - Сесия между приложението и драйвера на базата
- Пример за установяване на връзката през драйвера:

```
Connection con = DriverManager.  
    getConnection("url", "user", "pass");
```

- Синтаксис на URL в JDBC:  
 $\text{jdbc:<subprotocol>:<subname>}$
- JDBC-ODBC bridge драйвер, то адреса ще започва с "jdbc:odbc:"

# Изпълнение на заявки

- Обект **Statement** създава SQL заявките, които ще се изпратят към DBMS
  - За **SELECT** заявки се използва **executeQuery()**
  - За **INSERT/UPDATE/DELETE** заявки - **executeUpdate()**
  - Използва се активна инстанция на обект **Connection**

```
Connection dbCon = DriverManager.getConnection(  
    "jdbc:odbc:Library", "admin", "secret");  
Statement stmt = dbCon.createStatement();
```

# Изпълнение на заявки

- `executeQuery()` изпълнява предварително създадената SQL заявка
- Резултатът се връща като `ResultSet` обект

```
Connection dbCon =  
    DriverManager.getConnection(  
        "jdbc:odbc:Library", "admin", "secret");  
  
Statement stmt = dbCon.createStatement();  
  
ResultSet rs = stmt.executeQuery(  
    "SELECT first_name FROM employees");
```

# Изпълнение на заявки

- `executeUpdate()` изпълнява DML/DDL SQL заявки
  - Връща се резултат – броя на променените записи в базата

```
Statement stmt = dbCon.createStatement();
int rowsAffected = stmt.executeUpdate(
    "UPDATE employees SET salary = salary*1.2");
```

# Обработка на резултатите

- Обектът `ResultSet`
  - Поддържа курсор, сочещ към текущия запис
  - Има подходящи методи за извлечане на данните

```
ResultSet rs = stmt.executeQuery(  
    "SELECT last_name, salary FROM employees");  
while (rs.next()) {  
    String name = rs.getString("last_name");  
    double salary = rs.getDouble("salary");  
    System.out.println(name + " " + salary);  
}
```

# Затваряне на връзката

Унищожавайте Connection, Statement, и  
ResultSet обектите, за да се освободят  
ресурсите

```
try {
    Connection conn = ...;
    Statement stmt = ...;
    ResultSet rset = stmt.executeQuery(...);

    ...
} finally
    // clean up
    rset.close();
    stmt.close();
    conn.close();
}
```

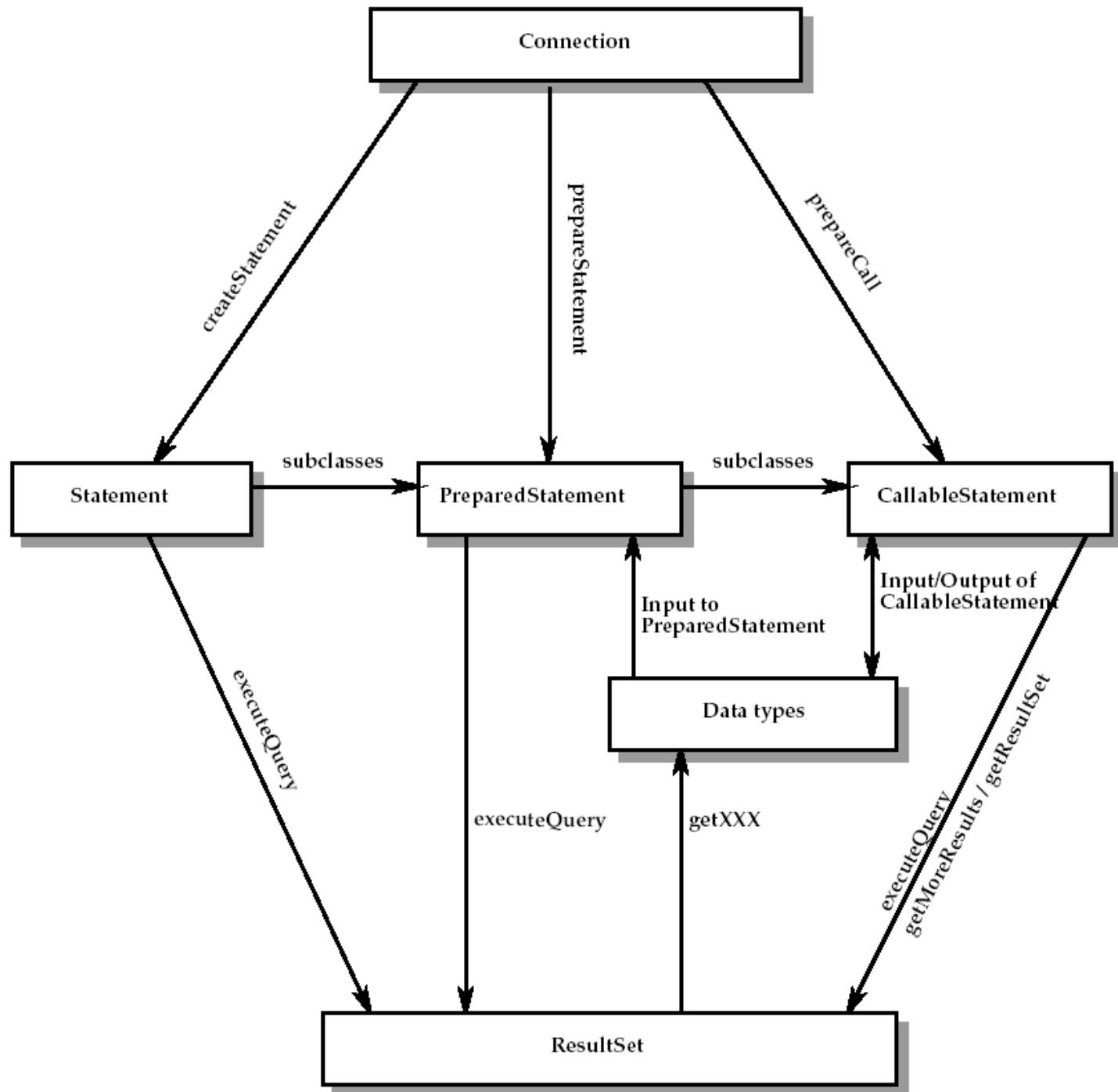
# Обект SQLException

SQL заявки могат да хвърлят  
java.sql.SQLException по време на  
изпълнение

```
try  {
    rset = stmt.executeQuery(
        "SELECT first_name, last_name FROM employee");
} catch (SQLException sqlex) {
    ... // Прихващаме SQL грешки
} finally {
    // Почистваме използваните ресурси
    try {
        if (rset != null) rset.close();
    } catch (SQLException sqlex) {
        ... // Игнорираме грешки при затваряне
    }
    ...
}
```

# **JDBC Заявки**

## **Интерфейсы Statement, PreparedStatement и CallableStatement**



# Statement

## 1. Създаваме Statement обект

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

## 2. Извикваме executeUpdate() за да изпълним заявката

```
int count = stmt.executeUpdate(sql_dml_statement);
```

- **Пример**

```
Statement stmt = conn.createStatement();
int rowsDeleted = stmt.executeUpdate("DELETE FROM
order_items WHERE order_id = 2354");
```

# Неизвестни заявки

## 1. Създаване Statement обект

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

## 2. Извикваме execute() за да изпълним

```
boolean result = stmt.execute(SQLstatement);
```

## 3. Обработваме резултата

```
if (result) { // било е select  
    ResultSet r = stmt.getResultSet(); ...  
}  
else { // било е update или от сорта  
    int count = stmt.getUpdateCount(); ...  
}
```

# Четене на автоматично генериран ключ

- За сървъри, които поддържат "auto increment" ключове
  - Например MS SQL Server, MySQL, ...
  - JDBC има достъп до генерираната стойност

```
// добавяме запис...
int rowCount = stmt.executeUpdate(
    "INSERT INTO Messages(Msg) VALUES ('Test')",
    Statement.RETURN_GENERATED_KEYS);

// ... и взимаме ключа
ResultSet rs = stmt.getGeneratedKeys();
rs.next();
long primaryKey = rs.getLong(1);
```

# Prepared Statements

- **PreparedStatement се използват за:**
  - Изпълнение на заявка с параметри
  - Многократно изпълнение на заявка

```
String insertSQL = "INSERT INTO employees (" +
    "first_name, last_name, salary) VALUES (?, ?, ?)" ;
PreparedStatement stmt =
    con.prepareStatement(insertSQL) ;
stmt.setString(1, "Ivan") ;
stmt.setString(2, "Ivanov") ;
stmt.setDouble(3, 25000.0) ;
stmt.executeUpdate() ;
```

# Prepared Statements

## Добри практики

- Използвайте винаги, когато имате заявки с параметри. Това е много по добре от употребата на оператор "+"
- Използвайте именовани параметри

```
String insertSQL = "INSERT INTO employees(" +  
    "first_name, last_name) VALUES(@fn, @ln)";  
PreparedStatement stmt =  
    con.prepareStatement(insertSQL);  
stmt.setString("@fn", "Ivan");  
stmt.setString("@ln", "Ivanov");  
stmt.executeUpdate();
```

# Примери

- Пример 1:

```
String updateString = "UPDATE COFFEES SET SALES=75 +  
                      "WHERE COF_NAME LIKE 'Colombian'" ;  
  
Statement stmt = conn.createStatement();  
stmt.executeUpdate(updateString);
```

- Пример 2:

```
PreparedStatement updateSales =  
    con.prepareStatement("UPDATE COFFEES SET" +  
    "SALES = ? WHERE COF_NAME LIKE ?") ;  
  
updateSales.setInt(1, 75) ;  
  
updateSales.setString(2, "Colombian") ;  
updateSales.executeUpdate() ;
```

# Заявки за извикване

- **Интерфейс CallableStatement**
  - За изпълнение на съхранени процедури
  - Приема входни параметри
  - Връща резултат
- **Пример:**

```
CallableStatement callStmt =  
    dbCon.prepareCall("call SP_Insert_Msg(?,?)");  
callStmt.setString(1, msgText);  
callStmt.registerOutParameter(2, Types.BIGINT);  
callStmt.executeUpdate();  
long id = callStmt.getLong(2);
```

# Работа с транзакции в JDBC

# Транзакции

- Транзакцията е набор от заявки, които трябва или да се изпълнят всичките, или нито една
- Обикновено транзакцията има вида:
  1. Отваряне на транзакция
  2. Изпълнение на няколко заявки
  3. Записване на транзакцията
- Ако някоя от заявките не се изпъни успешно цялата транзакция не се записва

# Транзакции

- Установяване на JDBC в режим с транзакции:
  - По подразбиране е режим Auto-Commit
  - Транзакциите се включват с `setAutoCommit(false)`
- В режим auto-commit всяка заявка се изпълнява в отделна транзакция
- При auto-commit off, промените няма да се запишат преди извикване на `commit()`
- Може да се върнем в auto-commit режим с `setAutoCommit(true)`

# Транзакции – Пример

- Ако не искаме промените да се запишат извикваме `rollback()`

```
dbCon.setAutoCommit(false);
try {
    Statement stmt = con.createStatement();
    stmt.executeUpdate("INSERT INTO Groups " +
        "VALUES (101, 'Administrators')");
    stmt.executeUpdate("INSERT INTO Users " +
        "VALUES (NULL, 'Mary', 101)");
    dbCon.commit();
} catch (Exception ex) {
    dbCon.rollback();
    throw ex;
}
```

# JDBC – добrite практики

# ResultSet Метаданни

- Клас ResultSetMetaData
  - Използва се за извличане на метаданните (колони, типове, ... ) от ResultSet обект

```
ResultSet rs = stmt.executeQuery(  
    "SELECT * FROM employees");  
ResultSetMetaData rsm = rs.getMetaData();  
int number = rsm.getColumnCount();  
for (int i=0; i<number; i++) {  
    System.out.println(rsm.getColumnName(i));  
}
```

# Почистване ненужните ресурси

- Изричното затваряне на обектите позволява на garbage collector-а да освободи максимално бързо ресурсите
- Унищожавайте Statement обектите веднага след прочитането им
- Използвайте try-finally за да сте сигурни, че ресурсите ще се освободят, дори и при грешка

# Правилната заявка

- Използвайте `PreparedStatement` когато заявката ще се изпълнява повече от веднъж, или когато се конструира с данни от потребителя
- Използвайте `CallableStatement` когато ви трябва резултат от много и сложни заявки за да получите един краен резултат

# Изпращане на няколко заявки наведнъж

- Изпращайте едновременно няколко заявки за да намалите извикванията към JDBC и да подобрите скоростта:

```
statement.addBatch ("sql_query1") ;  
statement.addBatch ("sql_query2") ;  
statement.addBatch ("sql_query3") ;  
  
statement.executeBatch () ;
```

- Можете да подобрите скоростта като повишите броя записи, които се връщат в един пакет от сървъра към драйвера

```
statement.setFetchSize (30) ;
```

# Оптимизирайте заявките

- **Лошо:**

```
Statement stmt = con.createStatement();  
  
ResultSet rs = stmt.executeQuery(  
    "SELECT * FROM EMPLOYEE WHERE ID=1");
```

- **Добре:**

```
Statement stmt = con.createStatement();  
  
ResultSet rs = stmt.executeQuery(  
    "SELECT SALARY FROM EMPLOYEE WHERE ID=1");
```

# Списък на JDBC драйвери (1)

Списък на драйверите за някои RDBMS :

- IBM DB2  
`jdbc:db2://<HOST>:<PORT>/<DB>`  
`COM.ibm.db2.jdbc.app.DB2Driver`
- JDBC-ODBC Bridge  
`jdbc:odbc:<DB>`  
`sun.jdbc.odbc.JdbcOdbcDriver`
- Microsoft SQL Server  
`jdbc:weblogic:mssqlserver4:<DB>@<HOST>:<PORT>`  
`weblogic.jdbc.mssqlserver4.Driver`
- Oracle Thin  
`jdbc:oracle:thin:@<HOST>:<PORT>:<SID>`  
`oracle.jdbc.driver.OracleDriver`
- PointBase Embedded Server  
`jdbc:pointbase://embedded[:<PORT>]/<DB>`  
`com.pointbase.jdbc.jdbcUniversalDriver`
- Cloudscape  
`jdbc:cloudscape:<DB>`  
`COM.cloudscape.core.JDBCDriver`
- Cloudscape RMI  
`jdbc:rmi://<HOST>:<PORT>/jdbc:cloudscape:<DB>`  
`RmiJdbc.RJDriver`
- Firebird (JCA/JDBC Driver)  
`jdbc:firebirdsql[://<HOST>[:<PORT>]]/<DB>`  
`org.firebirdsql.jdbc.FBDriver`

# Списък на JDBC драйвери (2)

- **IDS Server**  
`jdbc:ids://<HOST>:<PORT>/conn?dsn='<ODBC_DSN_NAME>'`  
`ids.sql.IDSDriver`
- **Informix Dynamic Server**  
`jdbc:informix-sqli://<HOST>:<PORT>/<DB>:INFORMIXSERVER=<SERVER_NAME>`  
`com.informix.jdbc.IfxDriver`
- **InstantDB (v3.13 and earlier)**  
`jdbc:idb:<DB>`  
`jdbc.idbDriver`
- **InstantDB (v3.14 and later)**  
`jdbc:idb:<DB>`  
`org.enhydra.instantdb.jdbc.idbDriver`
- **Interbase (InterClient Driver)**  
`jdbc:interbase://<HOST>/<DB>`  
`interbase.interclient.Driver`
- **Hypersonic SQL (v1.2 and earlier)**  
`jdbc:HypersonicSQL:<DB>`  
`hSql.hDriver`
- **Hypersonic SQL (v1.3 and later)**  
`jdbc:HypersonicSQL:<DB>`  
`org.hsqldb.jdbcDriver`
- **Microsoft SQL Server (JTurbo Driver)**  
`jdbc:JTurbo://<HOST>:<PORT>/<DB>`  
`com.ashna.jturbo.driver.Driver`
- **Microsoft SQL Server (Sprinta Driver)**  
`jdbc:inetdae:<HOST>:<PORT>?database=<DB>`  
`com.inet.tds.TdsDriver`

# Списък на JDBC драйвери (3)

- Microsoft SQL Server 2000 (Microsoft Driver)  
`jdbc:microsoft:sqlserver://<HOST>:<PORT>[;DatabaseName=<DB>]  
com.microsoft.jdbc.sqlserver.SQLServerDriver`
- MySQL (MM.MySQL Driver)  
`jdbc:mysql://<HOST>:<PORT>/<DB>  
org.gjt.mm.mysql.Driver`
- Oracle OCI 8i  
`jdbc:oracle:oci8:@<SID>  
oracle.jdbc.driver.OracleDriver`
- Oracle OCI 9i  
`jdbc:oracle:oci:@<SID>  
oracle.jdbc.driver.OracleDriver`
- PostgreSQL (v6.5 and earlier)  
`jdbc:postgresql://<HOST>:<PORT>/<DB>  
postgresql.Driver`
- PostgreSQL (v7.0 and later)  
`jdbc:postgresql://<HOST>:<PORT>/<DB>  
org.postgresql.Driver`
- Sybase (jConnect 4.2 and earlier)  
`jdbc:sybase:Tds:<HOST>:<PORT>  
com.sybase.jdbc.SybDriver`
- Sybase (jConnect 5.2)  
`jdbc:sybase:Tds:<HOST>:<PORT>  
com.sybase.jdbc2.jdbc.SybDriver`