# Databases in Java

## JDBC – Java DataBase Connectivity

- unified API for access to "table" data
- JDBC documentation: „... Access virtually any data source, from relational databases to spreadsheets and flat files“.
- we will use this to connect to SQLite and PostgreSQL
- https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html

## Basic steps on how to work with a database

1. establish a **connection**
2. create JDBC **statements**
3. execute **SQL queries**
4. process **ResultSet**
5. close the connection

https://docs.oracle.com/javase/tutorial/jdbc/basics/processingsqlstatements.html

# 1. Establish a connection

- **import java.sql.*;**
- **Load „vendor specific driver“**
    - Class.forName(“org.postgresql.Driver"); // PostgreSQL
      Class.forName("org.sqlite.JDBC"); // SQLite
- **Create a connection**
- Connection c = DriverManager.getConnection( "jdbc:postgresql://localhost:5432/username", username, password);
- // vytvorí sa TCP/IP spojenie medzi vašim programom a Postgresom

Connection c = DriverManager.getConnection("jdbc:sqlite:test.db");

  - // prístup k databáze sqlite sa rieši na úrovni práv k súboru

## 2. Create JDBC statements

- Statement stmt = con.createStatement() ;
- An object of type Statement will be created, which we will use to send SQL queries to the database.

## 3. Execute SQL queries

-
  ```
  String query = "Create table tblname "
      + "(id Integer not null, name VARCHAR(32), "
      + “marks Integer)";
  ```

  ```
  stmt.executeUpdate(query);
  ```

-
  ```
  String query = "Insert into tblname values“
      + "(123456789,’abc’,100)";
  ```

  ```
  stmt.executeUpdate(query);
  ```

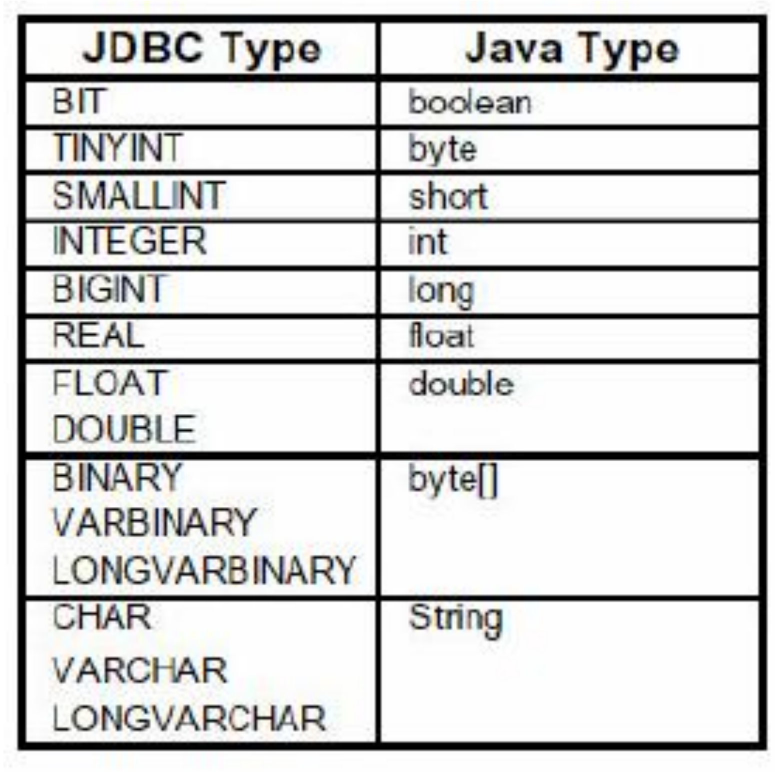
## 4. Process ResultSet

```
String query = "select * from Lehigh";

ResultSet rs = Stmt.executeQuery(query);


while (rs.next()) {
 int id= rs.getInt(“id");
 String name = rs.getString(“name");
 int marks = rs.getInt(“marks");
}
```

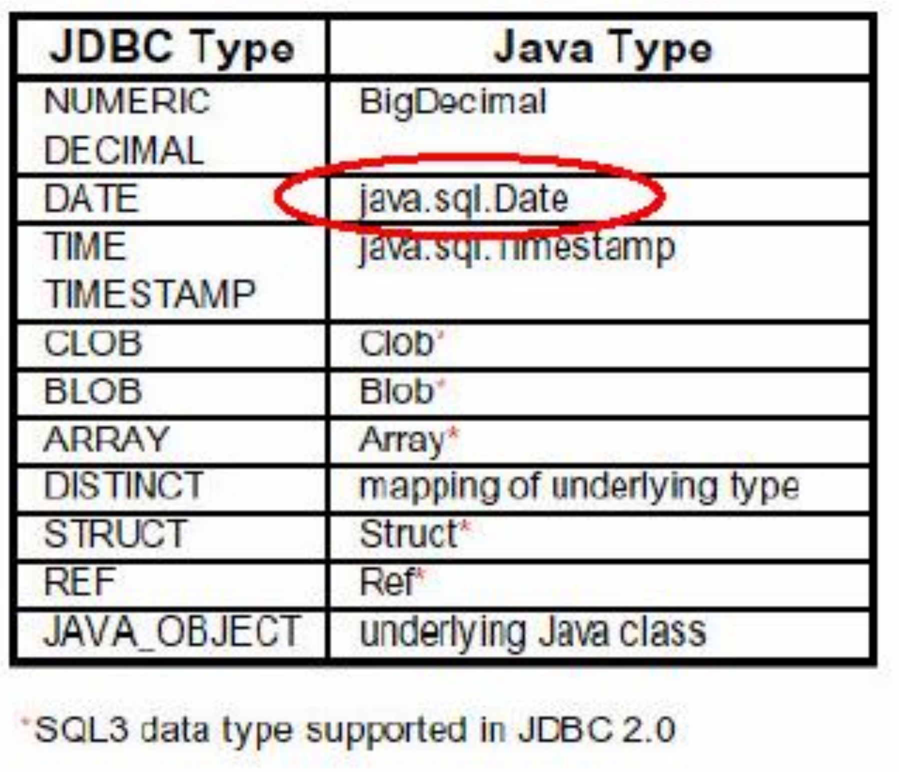
## 5. Close connection

- stmt.close();
- con.close();

# Typy JDBC - Java



| JDBC Type | Java Type |
|---|---|
| BIT | boolean |
| TINYINT | byte |
| SMALLINT | short |
| INTEGER | int |
| BIGINT | long |
| REAL | float |
| FLOAT DOUBLE | double |
| BINARY VARBINARY LONGVARBINARY | byte[] |
| CHAR VARCHAR LONGVARCHAR | String |



| JDBC Type | Java Type |
|---|---|
| NUMERIC DECIMAL | BigDecimal |
| DATE | java.sql.Date |
| TIME TIMESTAMP | java.sql.Timestamp |
| CLOB | Clob* |
| BLOB | Blob* |
| ARRAY | Array* |
| DISTINCT | mapping of underlying type |
| STRUCT | Struct* |
| REF | Ref* |
| JAVA_OBJECT | underlying Java class |

*SQL3 data type supported in JDBC 2.0

# Prepared statements

- https://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html

- Precompiled "parameterized" query.
  - PreparedStatement stmt = con.preparedStatement(
  - „SELECT * FROM student WHERE firstname=? AND lastname=?”);
  - stmt.setString(1,”jozef”);
  - stmt.setString(2,”mrkvicka”);
  - **ResultSet** rs = stmt.**executeQuery**();

## Prepared statements

- Advantages of prepared queries:
  - **Security – prevent SQL injection attacks**
  - **Efficiency –** the query is "parsed" only once

- **If the SQL query contains parameters that are controlled by the client, be sure to use the Prepared statement**

- If you do not use a prepared statement, you must ensure that the input escapes correctly.
  - JDBC does not have support for this (as it is specific to each DB system)
  - E.g. using "Dollar quoting" (a specialty of Postgres):
  - http://www.postgresql.org/docs/current/static/sql-syntax-lexical.html#SQL-SYNTAX-DOLLAR-QUOTING