

INTRODUCTION TO DATABASES

LESSON

WHAT IS DATA?

- ▶ In simple words data can be facts related to any object in consideration.
- ▶ For example your name, age, height, weight, etc are some data related to you.

WHAT IS DATABASE?

- ▶ A database is a collection of information that is organized so that it can be easily accessed, managed and updated.
- ▶ Data is organized into rows, columns and tables, and it is indexed to make it easier to find relevant information. Data gets updated, expanded and deleted as new information is added. Databases process workloads to create and update themselves, querying the data they contain and running applications against it.

WHAT IS A DATABASE MANAGEMENT SYSTEM?

- ▶ Database Management System (DBMS) is a collection of programs which enables its users to access database, manipulate data, reporting / representation of data.
- ▶ It also helps to control access to the database.
- ▶ Database Management Systems are not a new concept and as such had been first implemented in 1960s.

Types of DBMS



- ▶ **Hierarchical** - this type of DBMS employs the "parent-child" relationship of storing data. This type of DBMS is rarely used nowadays. Its structure is like a tree with nodes representing records and branches representing fields. The windows registry used in Windows XP is an example of a hierarchical database. Configuration settings are stored as tree structures with nodes.
- ▶ **Network DBMS** - this type of DBMS supports many-to many relations. This usually results in complex database structures. RDM Server is an example of a database management system that implements the network model.
- ▶ **Relational DBMS** - this type of DBMS defines database relationships in form of tables, also known as relations. Unlike network DBMS, RDBMS does not support many to many relationships. Relational DBMS usually have pre-defined data types that they can support. This is the most popular DBMS type in the market. Examples of relational database management systems include MySQL, Oracle, and Microsoft SQL Server database.
- ▶ **Object Oriented Relation DBMS** - this type supports storage of new data types. The data to be stored is in form of objects. The objects to be stored in the database have attributes (i.e. gender, age) and methods that define what to do with the data. PostgreSQL is an example of an object oriented relational DBMS.

WHAT IS SQL?

- ▶ Structured Query language (SQL) **pronounced as "S-Q-L" or sometimes as "See-Quel"** is actually the standard language for dealing with Relational Databases.
- ▶ SQL programming can be effectively used to insert, search, update, delete database records.
- ▶ Relational databases like MySQL Database, Oracle, Ms SQL server, Sybase, etc uses SQL !

SQL BASICS

The SQL CREATE TABLE Statement

The CREATE TABLE statement is used to create a new table in a database.

```
CREATE TABLE table_name (  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
    ....  
);
```

The column parameters specify the names of the columns of the table.

The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

Defines ID as primary key

Defines that id will be automatically incremented on insert

NOT_NULL defines
that column **cannot** be NULL

```
CREATE TABLE IF NOT EXISTS tasks (  
  id BIGINT NOT NULL AUTO_INCREMENT,  
  name VARCHAR(100) NOT NULL,  
  description VARCHAR(100) NULL,  
  created TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  PRIMARY KEY (id)  
)  
ENGINE = InnoDB  
AUTO_INCREMENT = 1;
```

Defines column that will contain time of insertion to table by default

Setting the Storage Engine

The SQL INSERT INTO Statement

The INSERT INTO statement is used to insert new records in a table.





The INSERT INTO Syntax

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

The SQL INSERT INTO Statement

```
INSERT INTO tasks (name, description) VALUES ('introduction to sql', 'level 1');
```

Result

	 id	 name	 description	 created
1	1003	introduction to sql	level 1	2019-03-19 13:35:23

The SQL SELECT Statement

The SELECT statement is used to select data from a database.

The data returned is stored in a result table, called the result-set.

```
SELECT column1, column2, ...  
FROM table_name;
```

Here, column1, column2, ... are the field names of the table you want to select data from.





If you want to select all the fields available in the table, use the following syntax:

```
SELECT * FROM table_name;
```

The SQL INSERT INTO Statement

```
SELECT * FROM tasks;
```

Result

	 id	 name	 description	 created
1	1010	introduction to sql	level 1	2019-03-19 13:48:59
2	1011	test sql 2	level 1	2019-03-19 13:49:00
3	1012	task 3	level 1	2019-03-19 13:49:00
4	1013	task 5	level 1	2019-03-19 13:49:00
5	1014	task 1	level 1	2019-03-19 13:49:00
6	1015	task 12	level 1	2019-03-19 13:49:00

The WHERE clause is used to filter records

```
SELECT * FROM tasks WHERE id=1010;
```

Result

	 id	 name	 description	 created
1	1010	introduction to sql	level 1	2019-03-19 13:48:59



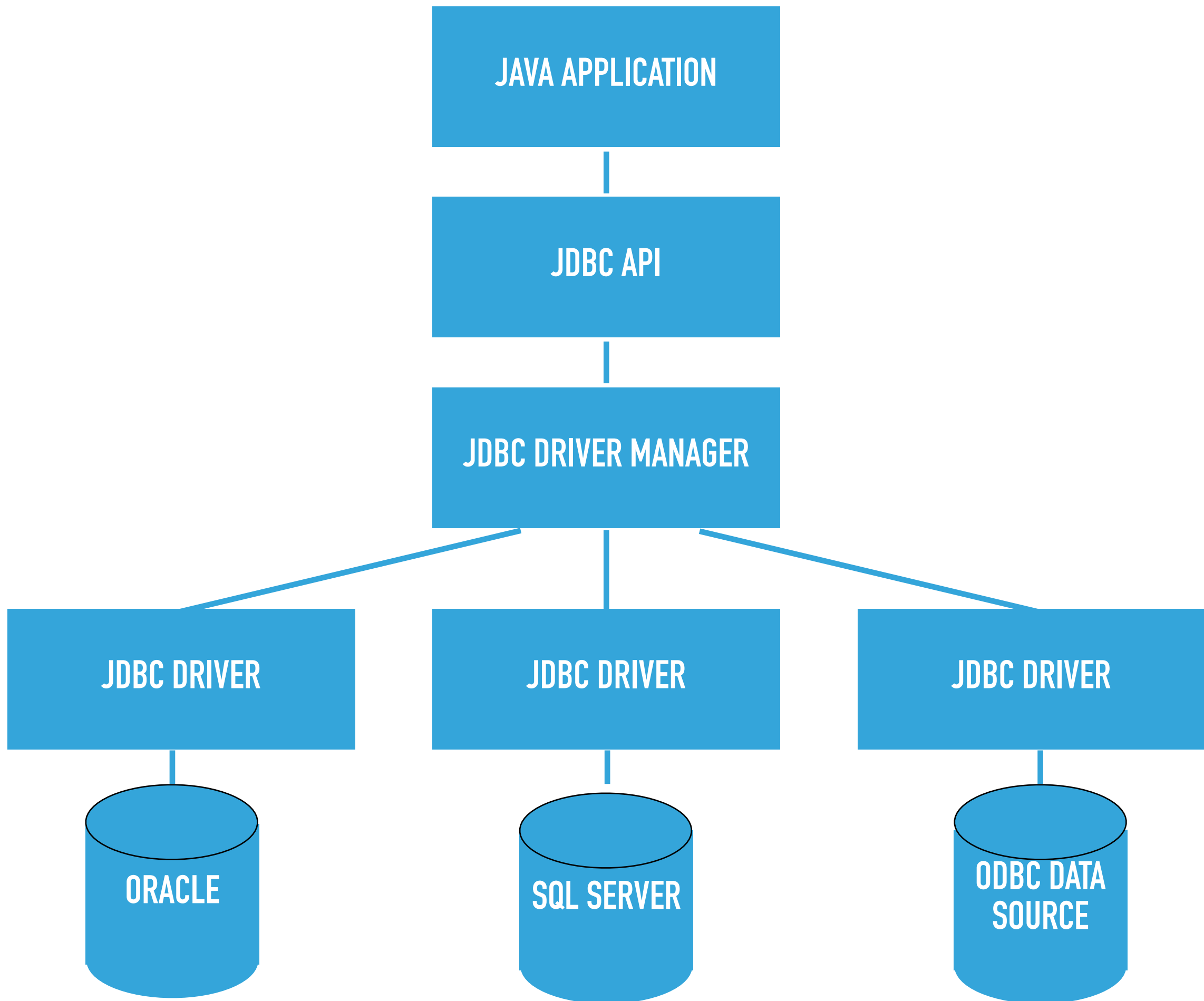
JDBC

JAVA DATABASE CONNECTIVITY

- ▶ JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:
 - ▶ JDBC-ODBC Bridge Driver
 - ▶ Native Driver
 - ▶ Network Protocol Driver
 - ▶ Thin Driver

JDBC ARCHITECTURE

- ▶ The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers:
 - ▶ **JDBC API:** This provides the application-to-JDBC Manager connection.
 - ▶ **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.
- ▶ The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.
- ▶ The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.



JDBC COMPONENTS

- ▶ **DriverManager:** This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.
- ▶ **Driver:** This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.
- ▶ **Connection:** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.

- ▶ **Statement:** You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
- ▶ **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
- ▶ **SQLException:** This class handles any errors that occur in a database application.

PREPARED STATEMENT

- ▶ JDBC PreparedStatement can be used when you plan to use the same SQL statement many times. It is used to handle precompiled query. If we want to execute same query with different values for more than one time then precompiled queries will reduce the no of compilations. Connection.prepareStatement() method can provide you PreparedStatement object. This object provides setXXX() methods to provide query values.

PREPARED STATEMENT EXAMPLE

```
Connection con = null;
PreparedStatement prSt = null;
try {
    Class.forName("oracle.jdbc.driver.OracleDriver");
    con = DriverManager.
        getConnection("jdbc:oracle:thin:@<hostname>:<port num>:<DB name>"
            , "user", "password");
    String query = "insert into emp(name,salary) values(?,?)";
    prSt = con.prepareStatement(query);
    prSt.setString(1, "John");
    prSt.setInt(2, 10000);
    //count will give you how many records got updated
    int count = prSt.executeUpdate();
    //Run the same query with different values
    prSt.setString(1, "Cric");
    prSt.setInt(2, 5000);
    count = prSt.executeUpdate();
} catch (ClassNotFoundException e) {
    e.printStackTrace();
} catch (SQLException e) {
    e.printStackTrace();
} finally{
    try{
        if(prSt != null) prSt.close();
        if(con != null) con.close();
    } catch (Exception ex){}
}
```

JdbcTemplate

JDBC TEMPLATE

- ▶ Spring **JdbcTemplate** is a powerful mechanism to connect to the database and execute SQL queries. It internally uses JDBC api, but eliminates a lot of problems of JDBC API.

PROBLEMS OF JDBC API

- ▶ We need to write a lot of code before and after executing the query, such as creating connection, statement, closing resultset, connection etc.
- ▶ We need to perform exception handling code on the database logic.
- ▶ We need to handle transaction.
- ▶ Repetition of all these codes from one to another database logic is a time consuming task.

ADVANTAGE OF SPRING JDBCTEMPLATE

- ▶ Spring JdbcTemplate eliminates all the above mentioned problems of JDBC API. It provides you methods to write the queries directly, so it saves a lot of work and time.

CONFIGURATION

application.properties

```
jdbc.url=jdbc:mysql://localhost:3306/todolist  
driverClass=com.mysql.jdbc.Driver  
database.user.name=root  
database.user.password=root
```

BEAN CONFIGURATION

```
@Configuration
@ComponentScan(basePackages = "com.javajava.todolist")
@PropertySource("classpath:application.properties")
public class AppConfig {

    @Value("${jdbc.url}")
    private String jdbcUrl;

    @Value("${driverClass}")
    private String driverClass;

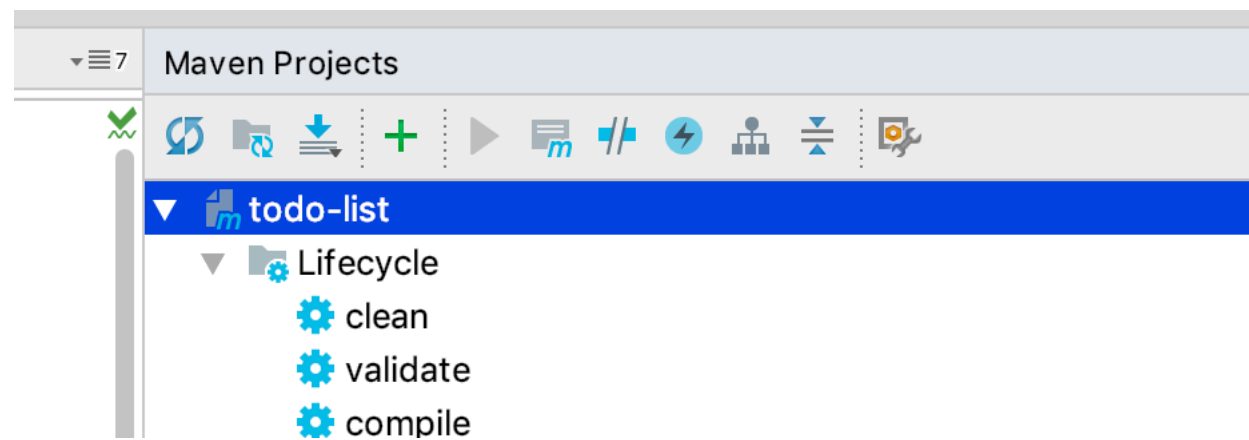
    @Value("${database.user.name}")
    private String userName;

    @Value("${database.user.password}")
    private String password;

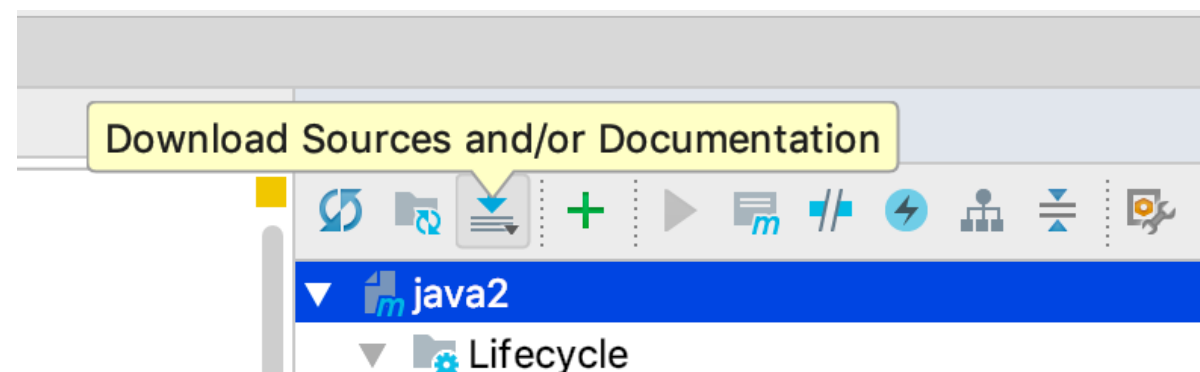
    @Bean
    public static PropertySourcesPlaceholderConfigurer propertySourcesPlaceholderConfigurer() {
        return new PropertySourcesPlaceholderConfigurer();
    }
}
```

SOURCES/DOCUMENTATION

1. Go to Maven Projects



2. Download Sources and/or Documentation



For example, click on

```
@PropertySource("classpath:application.properties")
```

```
@PropertySource(
```

Then press CTRL + B (command + B on MacOS)

Result:

```
/**
 * Annotation providing a convenient and declarative mechanism for adding a
 * {@link org.springframework.core.env.PropertySource PropertySource} to Spring's
 * {@link org.springframework.core.env.Environment Environment}. To be used in
 * conjunction with {@link Configuration} classes.
 *
 * <h3>Example usage</h3>
 *
 * <p>Given a file {@code app.properties} containing the key/value pair
 * {@code testbean.name=myTestBean}, the following {@code @Configuration} class
 * uses {@code @PropertySource} to contribute {@code app.properties} to the
 * {@code Environment}'s set of {@code PropertySources}.
```

```
...
```

@PropertySource annotation

Annotation providing a convenient and declarative mechanism for adding a `{@link org.springframework.core.env.PropertySource PropertySource}` to Spring's `{@link org.springframework.core.env.Environment Environment}`. To be used in conjunction with `{@link Configuration}` classes.

PropertySourcesPlaceholderConfigurer

class

Specialization of `{@link PlaceholderConfigurerSupport}` that resolves `${...}` placeholders within bean definition property values and `{@code @Value}` annotations against the current Spring `{@link Environment}` and its set of `{@link PropertySources}`.

DataSource configuration

- ▶ Java's `javax.sql.DataSource` interface provides a standard method of working with database connections. Traditionally, a 'DataSource' uses a URL along with some credentials to establish a database connection.

```
@Configuration
@ComponentScan(basePackages = "com.javajava.todolist")
@PropertySource("classpath:application.properties")
public class AppConfig {

    @Value("${jdbc.url}")
    private String jdbcUrl;

    @Value("${driverClass}")
    private String driverClass;

    @Value("${database.user.name}")
    private String userName;

    @Value("${database.user.password}")
    private String password;

    @Bean
    public static PropertySourcesPlaceholderConfigurer propertySourcesPlaceholderConfigurer() {
        return new PropertySourcesPlaceholderConfigurer();
    }

    @Bean
    public DataSource dataSource() {
        BasicDataSource dataSource = new BasicDataSource();
        dataSource.setUrl(jdbcUrl);
        dataSource.setDriverClassName(driverClass);
        dataSource.setUsername(userName);
        dataSource.setPassword(password);
        return dataSource;
    }

    @Bean
    public JdbcTemplate jdbcTemplate() {
        return new JdbcTemplate(dataSource());
    }
}
```

JdbcTemplate query example

```
public Optional<Task> findTaskById(Long id) {  
    String query = "select * from tasks where id=" + id;  
  
    List<Task> tasks = jdbcTemplate.query(query,  
        new BeanPropertyRowMapper(Task.class));  
  
    if (!tasks.isEmpty()) {  
        return Optional.ofNullable(tasks.get(0));  
    }  
    return Optional.empty();  
}
```

BeanPropertyRowMapper

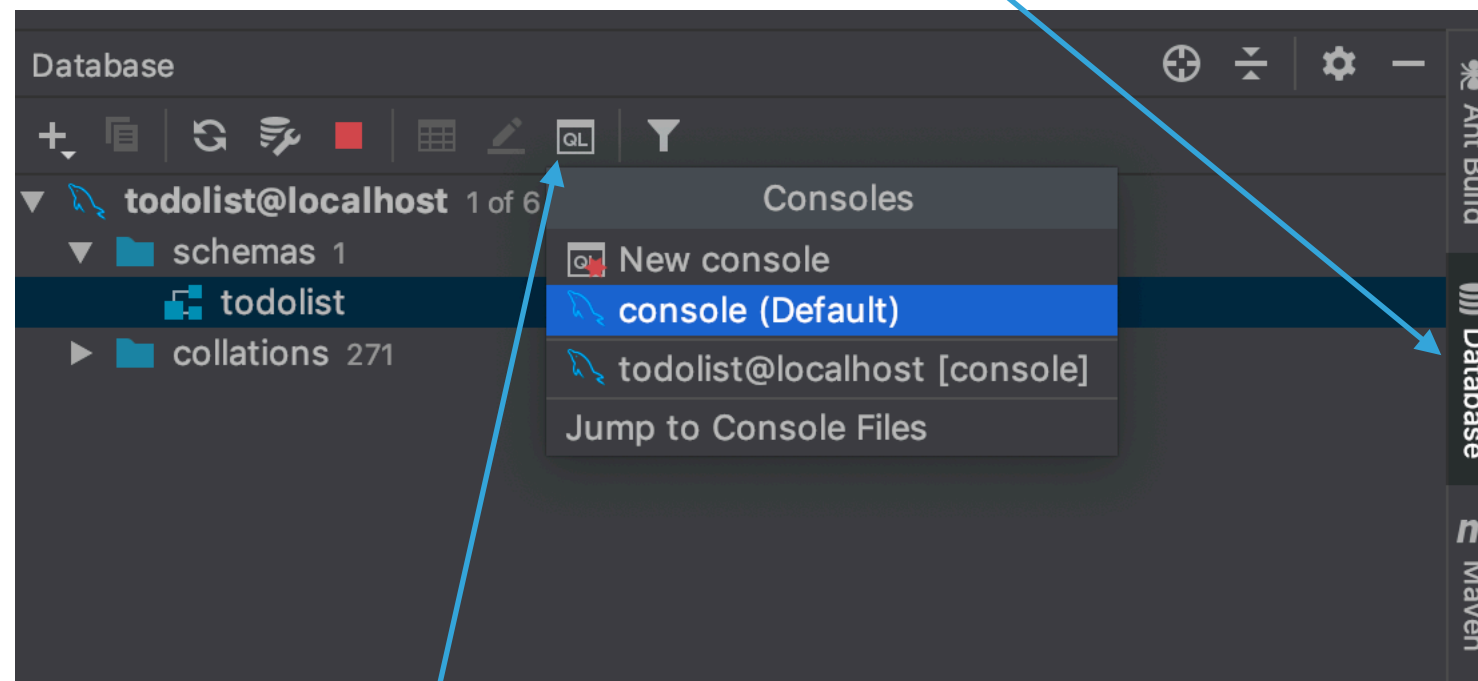
RowMapper implementation that converts a row into a new instance of the specified mapped target class. The mapped target class must be a top-level class and it must have a default or no-arg constructor.

Column values are mapped based on matching the column name as obtained from result set meta-data to public setters for the corresponding properties. The names are matched either directly or by transforming a name separating the parts with underscores to the same name using "camel" case.

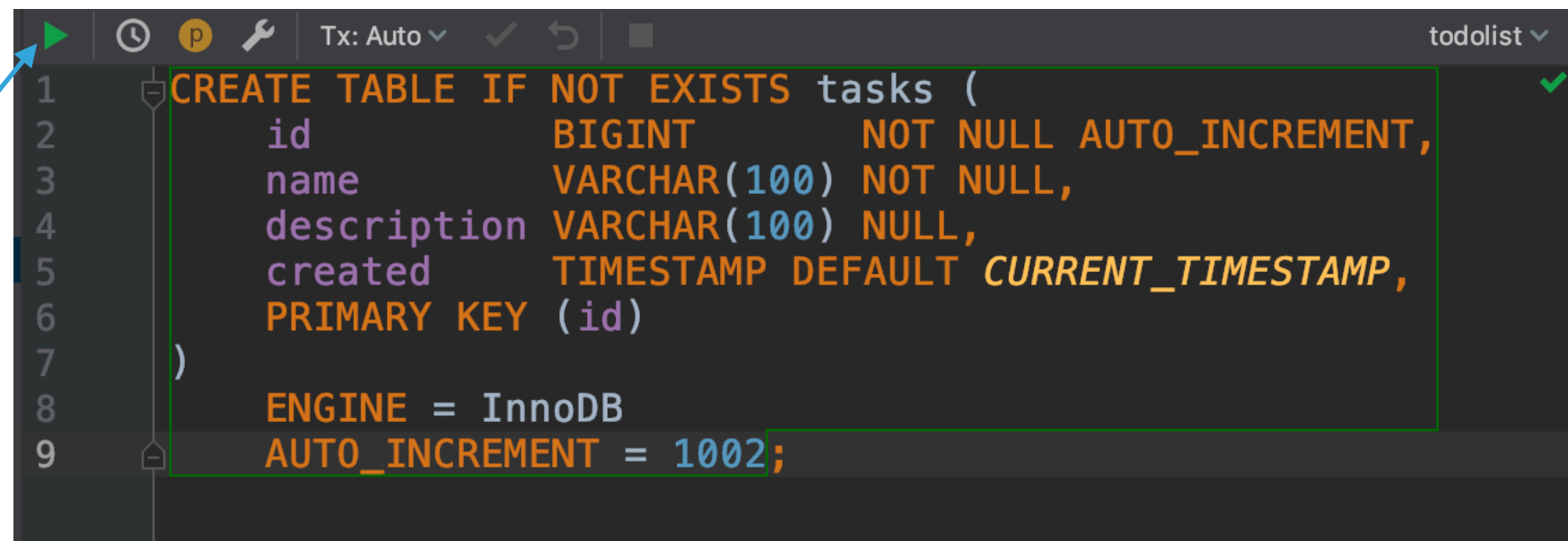
SQL CONSOLE

IntelliJ IDEA Ultimate SQL console example

Step 1. Open "Database" tab



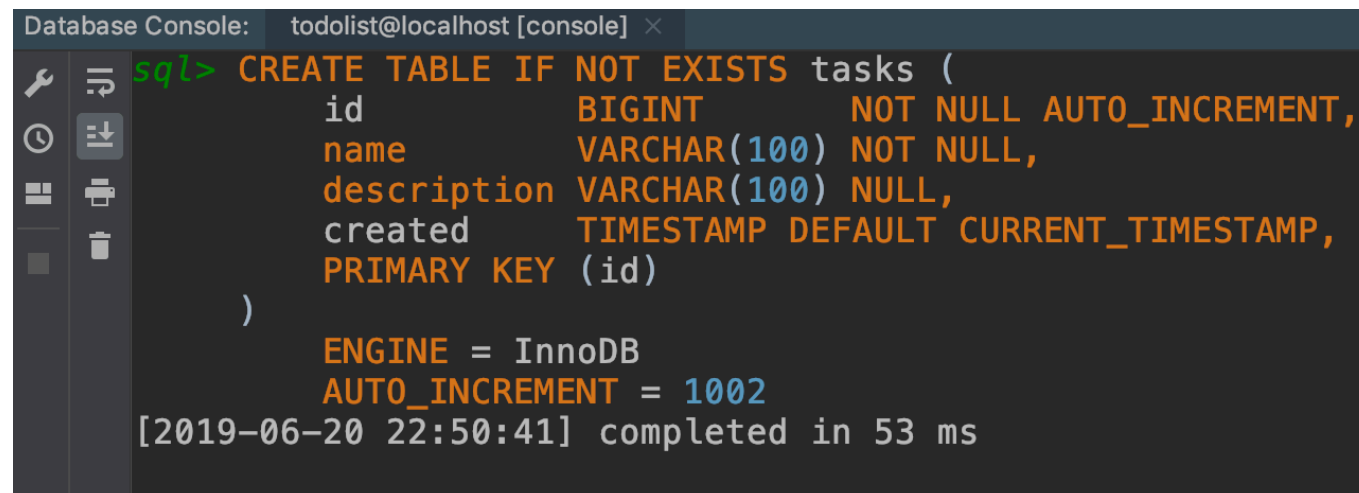
Step 2. Open "Consoles" menu



```
1 CREATE TABLE IF NOT EXISTS tasks (  
2     id          BIGINT          NOT NULL AUTO_INCREMENT,  
3     name        VARCHAR(100) NOT NULL,  
4     description  VARCHAR(100) NULL,  
5     created      TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
6     PRIMARY KEY (id)  
7 )  
8 ENGINE = InnoDB  
9 AUTO_INCREMENT = 1002;
```

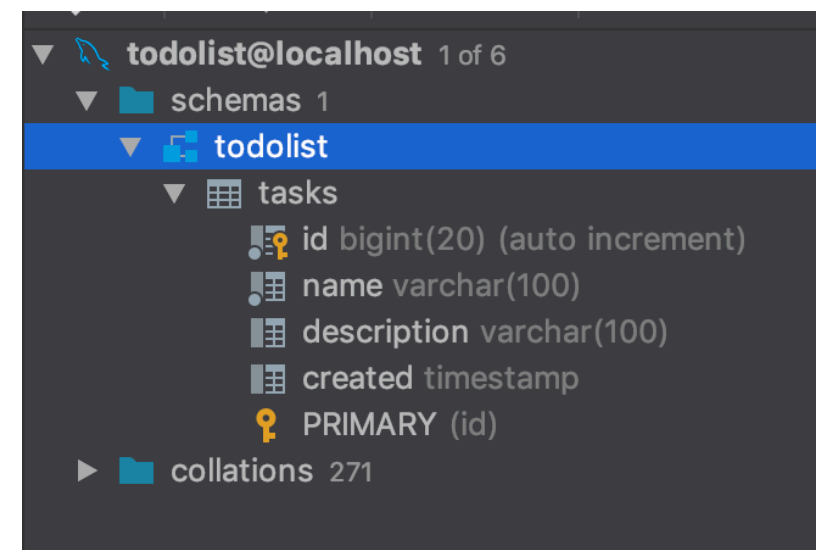
Step 3. Enter your query and press "Execute"

Console output (result):



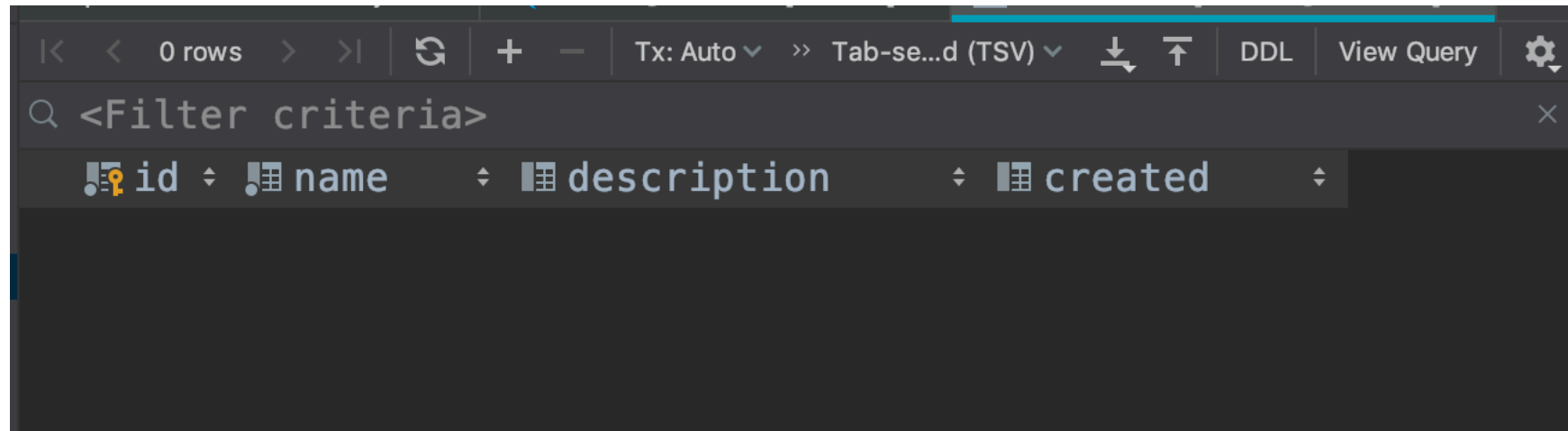
```
Database Console: todolist@localhost [console] x  
sql> CREATE TABLE IF NOT EXISTS tasks (  
    id          BIGINT          NOT NULL AUTO_INCREMENT,  
    name        VARCHAR(100) NOT NULL,  
    description  VARCHAR(100) NULL,  
    created      TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    PRIMARY KEY (id)  
)  
ENGINE = InnoDB  
AUTO_INCREMENT = 1002  
[2019-06-20 22:50:41] completed in 53 ms
```

New table added:

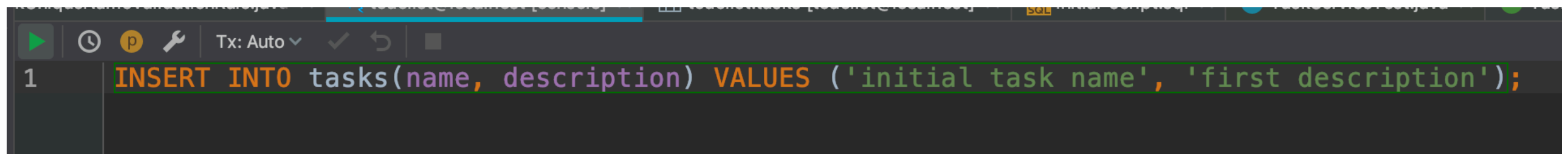


```
▼ todolist@localhost 1 of 6  
  ▼ schemas 1  
    ▼ todolist  
      ▼ tasks  
        id bigint(20) (auto increment)  
        name varchar(100)  
        description varchar(100)  
        created timestamp  
        PRIMARY (id)  
      ► collations 271
```


From now you have empty table







Now you can execute any sql query. For example insert:



Console output:

```
sql> INSERT INTO tasks(name, description) VALUES ('initial task name', 'first description')
[2019-06-20 22:54:00] 1 row affected in 19 ms
```


New row inserted! Success!

	 id ▾	 name ▾	 description ▾	 created ▾
1	1002	initial task name	first description	2019-06-20 19:54:00

REFERENCES

- ▶ <https://www.w3schools.com/sql/default.asp>
- ▶ <https://dev.mysql.com/downloads/mysql/>
- ▶ <https://www.baeldung.com/spring-jdbc-jdbctemplate>
- ▶ <https://docs.spring.io/spring-boot/docs/current/reference/html/howto-data-access.html>