JDBC
Java Database Connectivity

What's a JDBC Driver?

- To use a particular data source from an application, you'll need a JDBC driver for that data source.
- A driver is simply a Java library, containing classes that implement the JDBC API.
- Drivers are usually provided by the database vendor, either as a library jar, or a java module, which we can import into our application.
- The current version of the JDBC API, is JDBC 4.3.

JDBC Drivers

- J**ava Database Connectivity (JDBC)** is an application programming interface (API) for the programming language Java, which defines how a client may access any kind of tabular data, especially a relational database.
- **JDBC Drivers** uses JDBC APIs which was developed by Sun Microsystem, but now this is a part of Oracle.
- There are 4 types of JDBC drivers. It is part of the Java Standard Edition platform, from Oracle Corporation. It acts as a middle-layer interface between Java applications and databases.
  - Type-1 driver or JDBC-ODBC bridge driver
  - Type-2 driver or Native-API driver
  - Type-3 driver or Network Protocol driver
  - Type-4 driver or Thin driver

3

JDBC Connection URLs by some vendors

- MySQL - "jdbc:mysql://localhost:3306/music"
- PostgreSQL - "jdbc:postgresql://localhost:5432/music"
- Oracle - "jdbc:oracle:thin:@localhost:1521/music"
- Microsoft SQL Server - "jdbc:sqlserver://localhost:1433;databaseName=music"
- SQLite - "jdbc:sqlite:music.db"

4

Connecting to a Database

```
Class.forName("com.mysql.cj.jdbc.Driver");

Connection con = DriverManager
                        .getConnection("jdbc:mysql://localhost:3306/myDb", "user1", "pass"))
```

Types of Statements in JDBC

- The statement interface is used to create SQL basic statements in Java it provides methods to execute queries with the database.
- There are different types of statements that are used in JDBC as follows:
  - Create Statement
  - Prepared Statement
  - Callable Statement

Statement

```
con.createStatement();
String tableSql = "CREATE TABLE IF NOT EXISTS employees"
  + "(emp_id int PRIMARY KEY AUTO_INCREMENT, name varchar(30),"
  + "position varchar(30), salary double)";
stmt.execute(tableSql);


String selectSql = "SELECT * FROM employees";
try (ResultSet resultSet = stmt.executeQuery(selectSql)) {
    // use resultSet here
}
```

PreparedStatement

```
String updatePositionSql = "UPDATE employees SET position=? WHERE emp_id=?";
try (PreparedStatement pstmt = con.prepareStatement(updatePositionSql)) {
    // use pstmt here
}
pstmt.setString(1, "lead developer");
pstmt.setInt(2, 1);
int rowsAffected = pstmt.executeUpdate();
```

CallableStatement

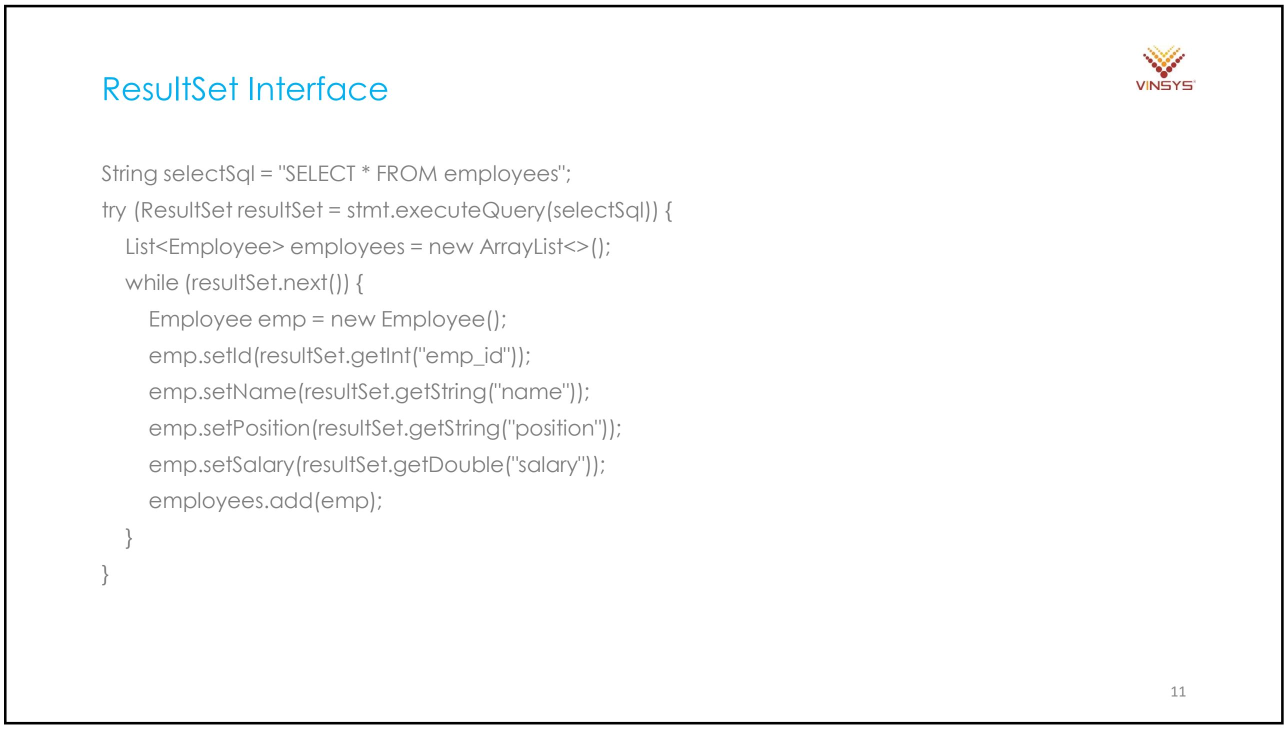
```
String preparedSql = "{call insertEmployee(?,?,?,?)}";
try (CallableStatement cstmt = con.prepareCall(preparedSql)) {
    // use cstmt here
}

cstmt.setString(2, "ana");
cstmt.setString(3, "tester");
cstmt.setDouble(4, 2000);

cstmt.registerOutParameter(1, Types.INTEGER);
cstmt.execute();
int new_id = cstmt.getInt(1);
```

Stored Procedure (MySQL)

```
CREATE PROCEDURE insertEmployee(OUT emp_id int,
                    IN emp_name varchar(30), IN position varchar(30), IN salary double)
BEGIN
INSERT INTO employees(name, position,salary) VALUES (emp_name,position,salary);
SET emp_id = LAST_INSERT_ID();
END
```



ResultSet Interface

```
String selectSql = "SELECT * FROM employees";
try (ResultSet resultSet = stmt.executeQuery(selectSql)) {
    List<Employee> employees = new ArrayList<>();
    while (resultSet.next()) {
        Employee emp = new Employee();
        emp.setId(resultSet.getInt("emp_id"));
        emp.setName(resultSet.getString("name"));
        emp.setPosition(resultSet.getString("position"));
        emp.setSalary(resultSet.getDouble("salary"));
        employees.add(emp);
    }
}
```