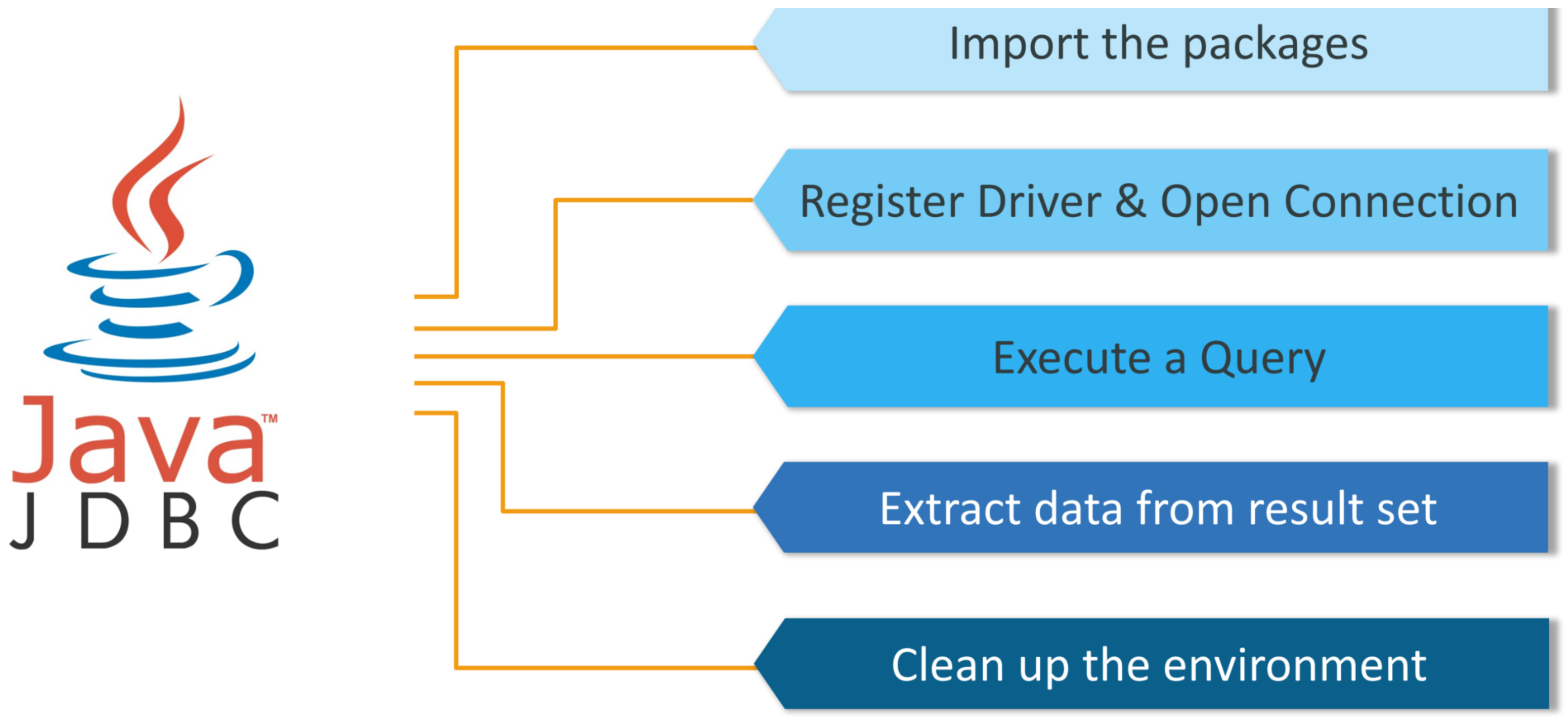
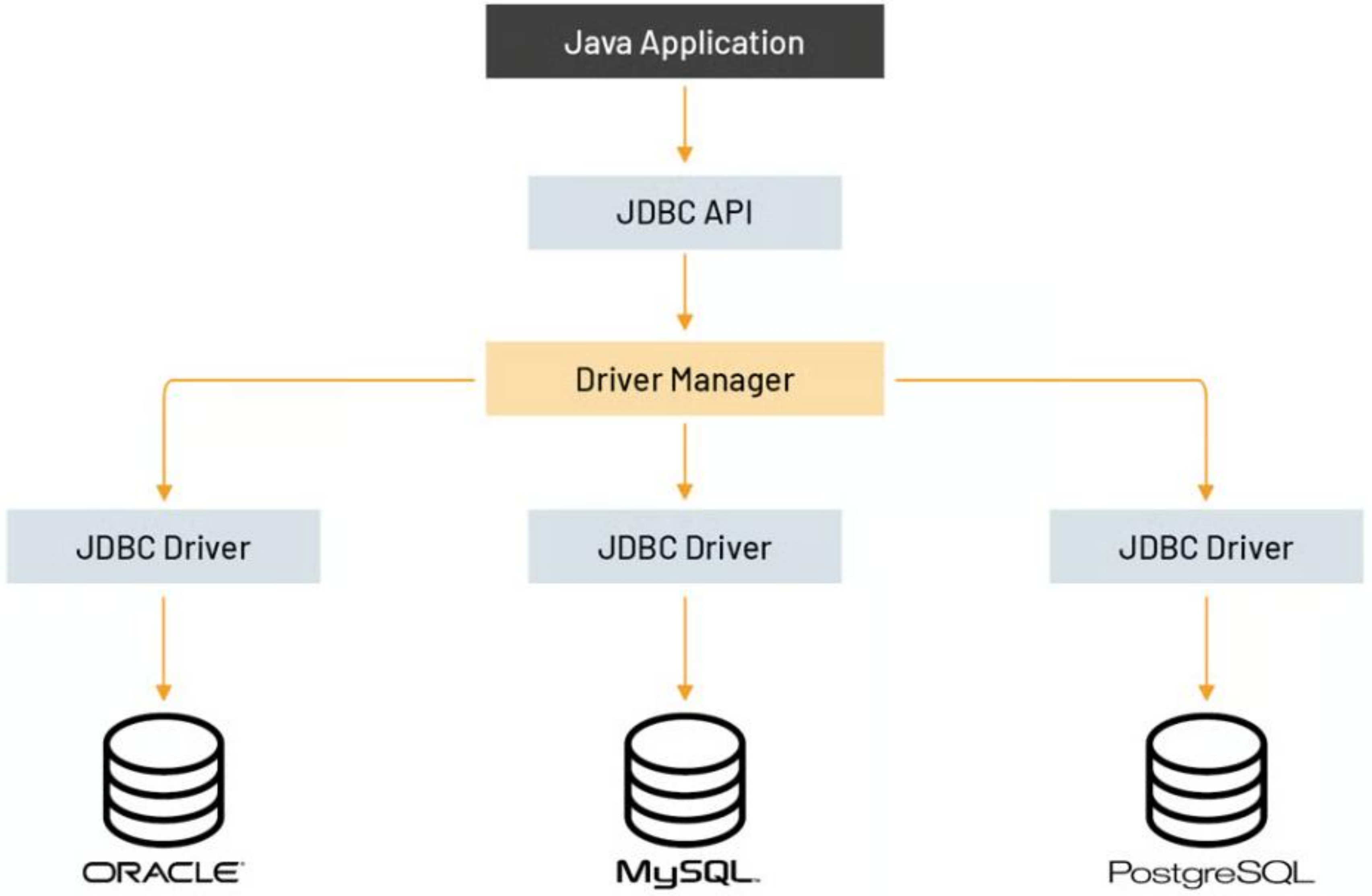


Java Database Connectivity

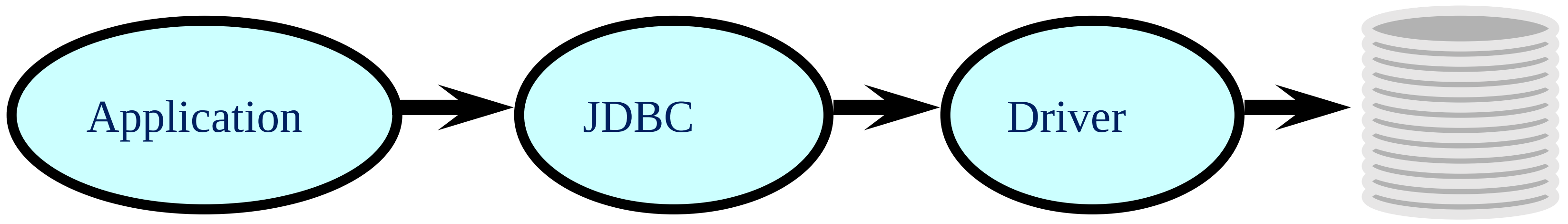
Introduction to JDBC

- JDBC – Java Database Connectivity
- **JDBC** is used for accessing databases from Java applications
- Information is transferred from relations to objects and vice-versa

JDBC Architecture



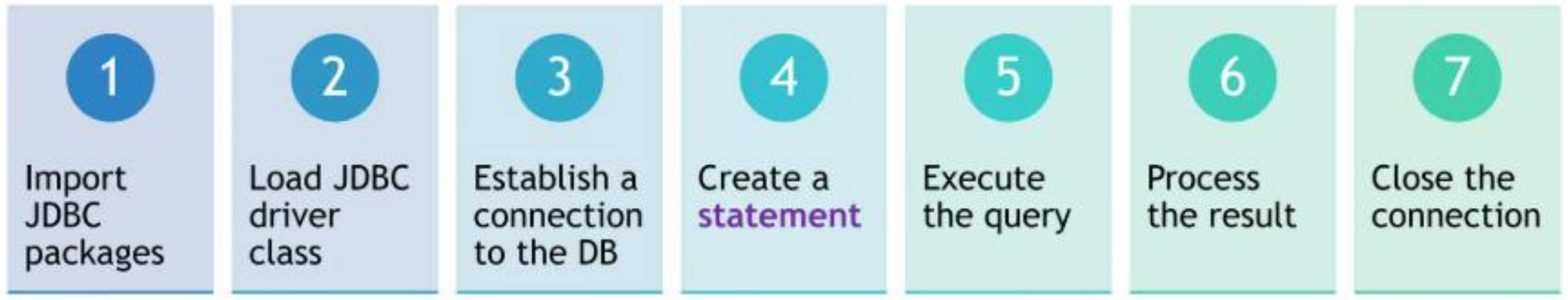
JDBC Architecture



- Java code calls JDBC library
- JDBC loads a *driver*
- Driver talks to a particular database
- An application can work with several databases by using all corresponding drivers
- Ideal: can change database engines *without changing any application code* (not always in practice)

Seven Steps

- Load the driver
- Define the connection URL
- Establish the connection
- Create a **Statement** object
- Execute a query using the **Statement**
- Process the result
- Close the connection



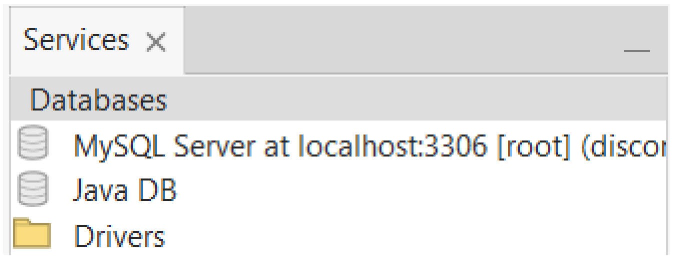
Loading the Driver

- We can register the driver indirectly using the statement

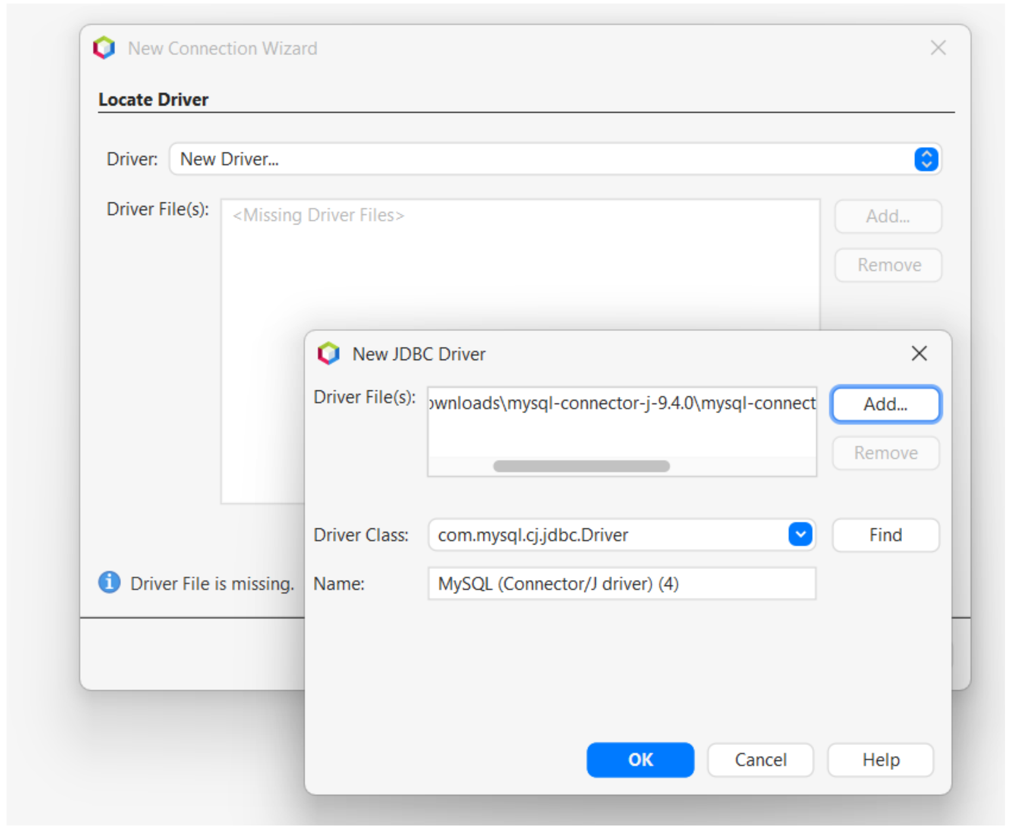
  Class.forName("com.mysql.jdbc.Driver");
- Class.forName loads the specified class
- When mysqlDriver is loaded, it automatically
  - creates an instance of itself
  - registers this instance with the DriverManager

- Hence, the driver class can be given as an argument of the application

1



2


New Connection Wizard

Locate Driver

Driver: New Driver...

Driver File(s): <Missing Driver Files>

Driver File is missing.

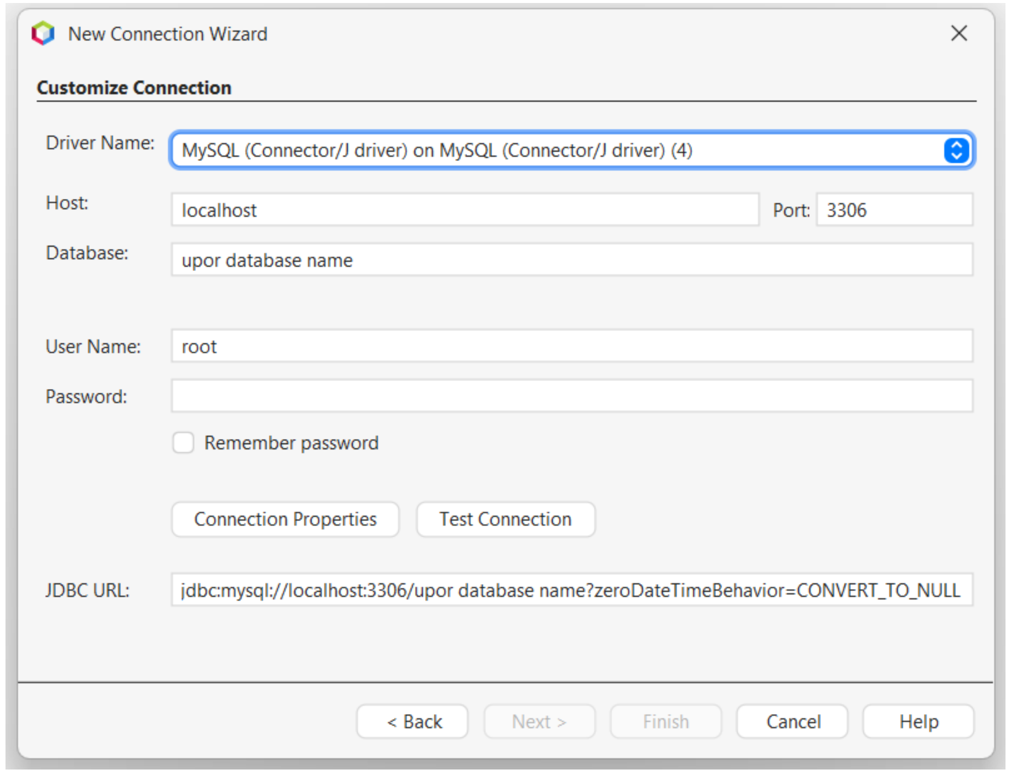
New JDBC Driver

Driver File(s): ›wnloads\mysql-connector-j-9.4.0\mysql-connect

Driver Class: com.mysql.cj.jdbc.Driver

Name: MySQL (Connector/J driver) (4)

3


New Connection Wizard

Customize Connection

Driver Name: MySQL (Connector/J driver) on MySQL (Connector/J driver) (4)

Host: localhost Port: 3306

Database: upor database name

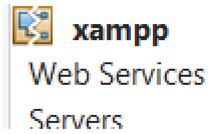
User Name: root

Password:

Remember password

JDBC URL: jdbc:mysql://localhost:3306/upor database name?zeroDateTimeBehavior=CONVERT_TO_NULL

4

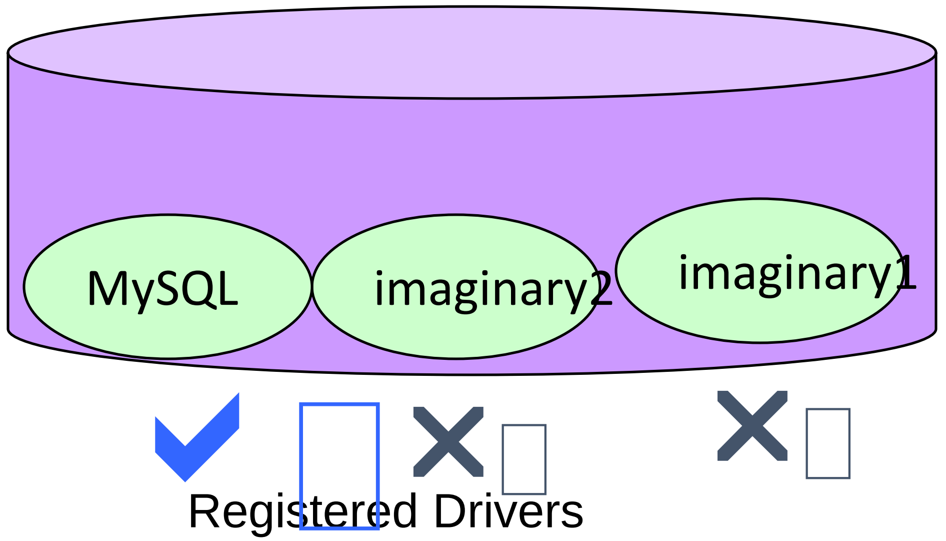


Connecting to the Database

- Every database is identified by a URL
- Given a URL, DriverManager looks for the driver that can talk to the corresponding database
- DriverManager tries all registered drivers, until a suitable one is found

Connecting to the Database

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



```
String url = "jdbc:mysql://localhost:3306/students";
String user = "root";
String password = "";
```

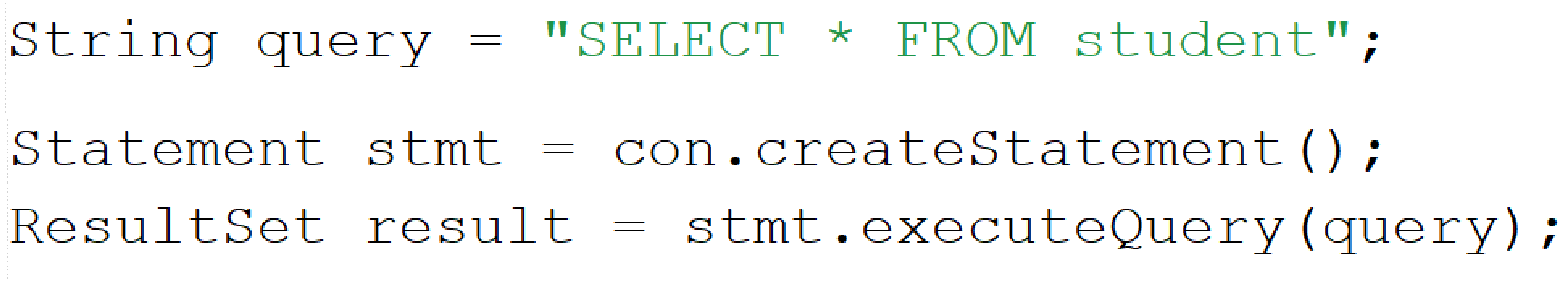
Interaction with the Database

- We use Statement objects in order to
  - Query the database
  - Update the database

- Three different interfaces are used:

  Statement, PreparedStatement, CallableStatement
- All are interfaces, hence cannot be instantiated
- They are created by the Connection

Querying with Statement



```
String query = "SELECT * FROM student";

Statement stmt = con.createStatement();
ResultSet result = stmt.executeQuery(query);
```

- The executeQuery method returns a ResultSet object representing the query result.
  - Will be discussed later…

About Prepared Statements

- Prepared Statements are used for queries that are executed many times
- They are parsed (compiled) by the DBMS only once
- Column values can be set after compilation
- Instead of values, use ‘?’
- Hence, Prepared Statements can be though of as statements that contain placeholders to be substituted later with actual values

Querying with PreparedStatement

```
String queryStr =
        "SELECT * FROM employee " +
        "WHERE superssn= ? and salary > ?";


PreparedStatement   pstmt =
        con.prepareStatement(queryStr);


pstmt.setString(1, "333445555");
pstmt.setInt(2, 26000);


ResultSet rs = pstmt.executeQuery();
```

Updating with PreparedStatement

```
String deleteStr =
        “DELETE FROM employee " +
        "WHERE superssn = ? and salary > ?";

PreparedStatement pstmt =       con.prepareStatement(deleteStr);

pstmt.setString(1, "333445555");
pstmt.setDouble(2, 26000);

int delnum = pstmt.executeUpdate();
```

ResultSet

- ResultSet objects provide access to the tables generated as results of executing a Statement queries
- Only one ResultSet per Statement can be open at the same time!
- The table rows are retrieved in sequence
  - A ResultSet maintains a cursor pointing to its current row
  - The next() method moves the cursor to the next row

Cleaning Up After Yourself

- Remember to close the Connections, Statements, Prepared Statements and Result Sets

```
con.close();
stmt.close();
pstmt.close();
rs.close()
```

Dealing With Exceptions

- An SQLException is actually a list of exceptions

```
catch (SQLException e) {
   while (e != null) {
        System.out.println(e.getSQLState());
        System.out.println(e.getMessage());
        System.out.println(e.getErrorCode());
        e = e.getNextException();
   }
}
```

Mapping Java Types to SQL Types

SQL type	Java Type
CHAR, VARCHAR, LONGVARCHAR	String
NUMERIC, DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT, DOUBLE	double
BINARY, VARBINARY, LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Thank You