Creating Your Database Using JDBC

You Already Know

Course(s):

Fundamentals of SQL

Recap

- Explain what are relational databases
  - What are databases?
  - Tables and relations

- Demonstrate SQL Queries
  - What is querying?
  - DML, DDL, and more
  - Common query tool

- Filtering results using Queries
  - Creating a filter condition
  - Applying multiple filter conditions

Recap

- Consolidate and group your data using queries
  - Unions and other multiset consolidations
  - The group by clause

- Join tables
  - Inner joins, outer joins, and self joins

- Manipulate data
  - Insert statement
  - Update statement
  - Delete statement

A Day in the Life of a Test Engineer

Joe is now working on a database management system. He is asked to develop a relational database for an e-commerce company.

To make the user experience better, Joe must write a program where the particular product details are fetched from the database and displayed on the dashboard after the user enters the product ID.

In this lesson, we will learn how to solve this real-world scenario to help Joe complete his task effectively and quickly.

Learning Objectives

By the end of this lesson, you will be able to:

- Explain JDBC
- Illustrate a connection in JDBC
- Describe statement and its types
- Demonstrate resultset and its types
- Describe stored procedures
- Explain exception handling in stored procedures
- Describe database creation, selection, and dropping
- Demonstrate insertion, updation, and deletion of database records

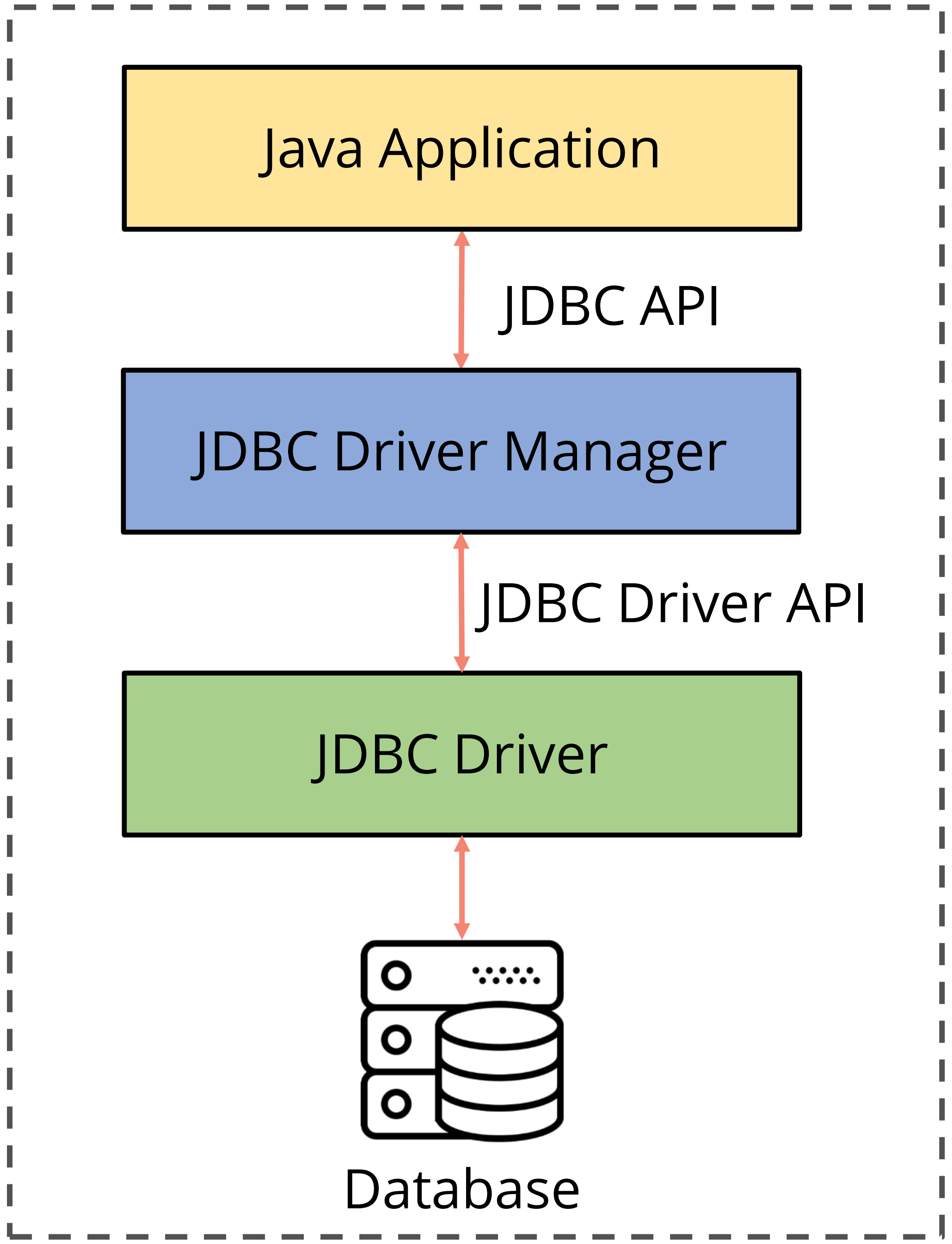
FULL STACK

Overview of JDBC

Overview of JDBC

Java Database Connectivity (JDBC) is a Java-based and database-agnostic data access technology that defines how a client may access a database.

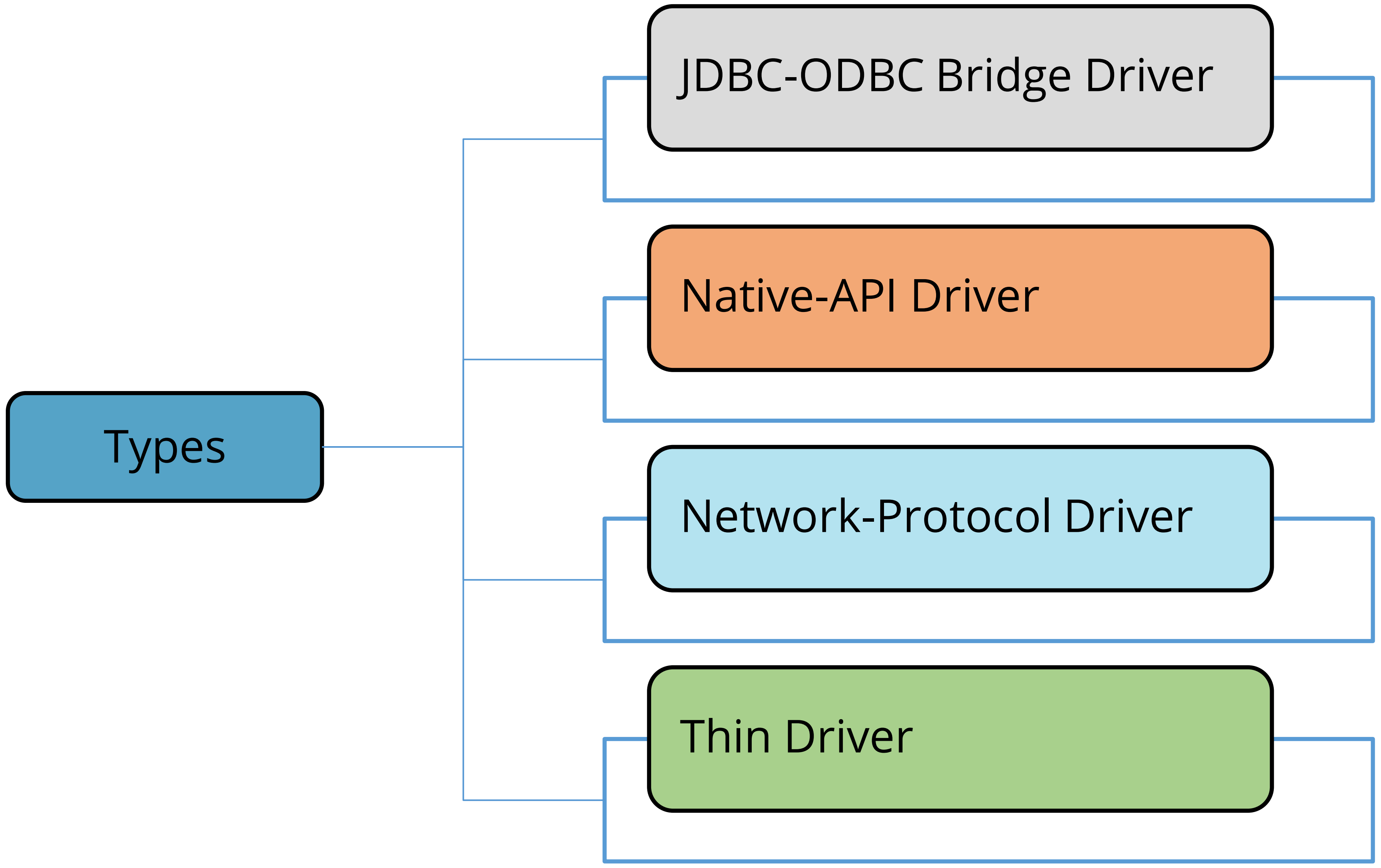
JDBC Architecture



Latest JDBC version: JDBC 4.3

JDBC Driver and Types

A JDBC driver is a software component that enables a Java application to interact with a database.



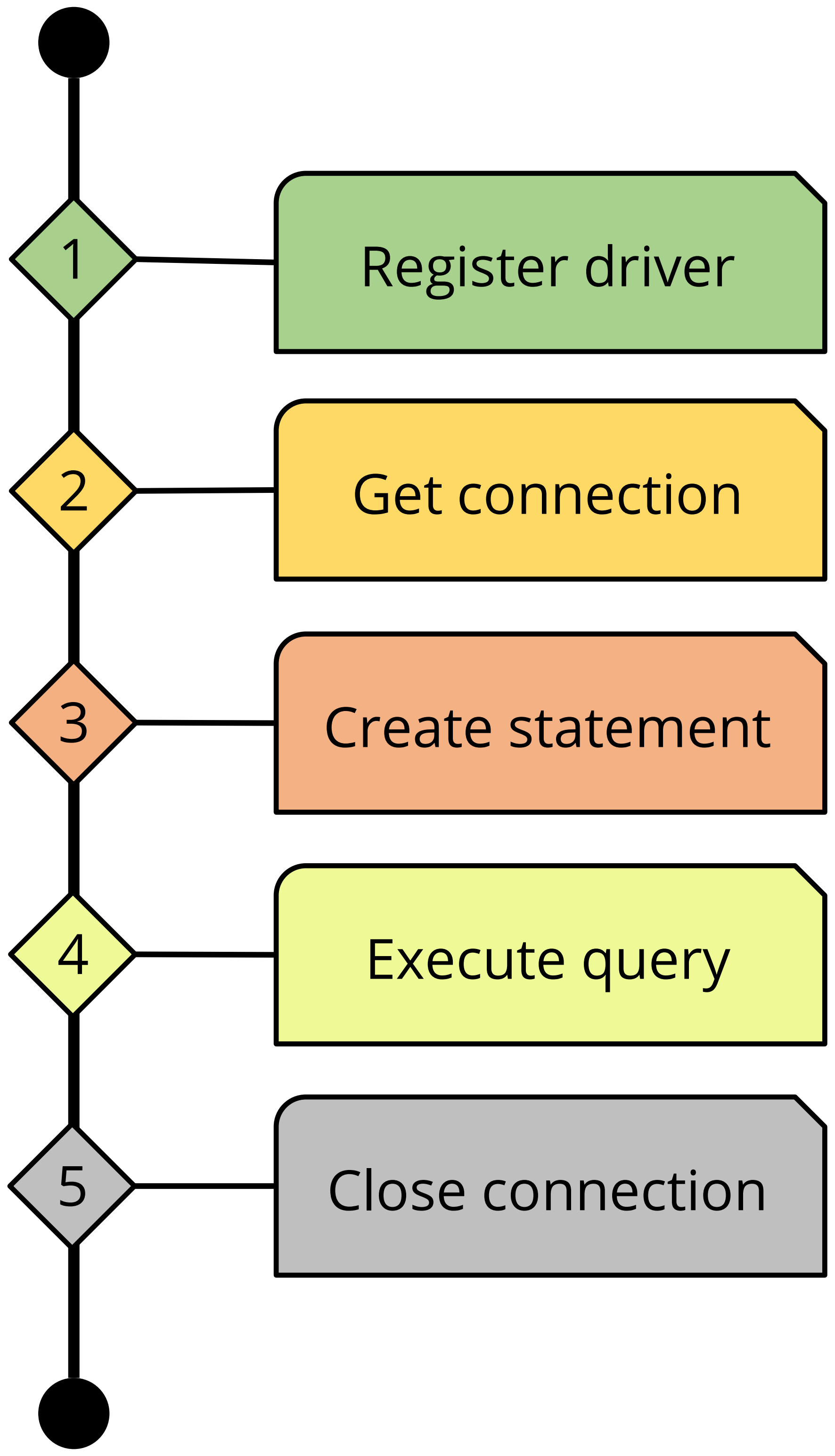
JDBC API

A JDBC API provides data access from the Java programming language. JDBC API includes java.sql and javax.sql packages. JDBC API is implemented through a JDBC driver.

JDBC API Objects

DataSource	Establishes connections
Connection	Controls connection to the database
Statement, PreparedStatement, and CallableStatement	Execute SQL queries
ResultSet	Holds the results of a query

JDBC Connectivity



Set up a JDBC Environment

Duration: 25 min.

Problem Statement:

Set up a JDBC environment.

Assisted Practice: Guidelines

Steps to set up a JDBC environment:

1. Create a dynamic web project to set up a JDBC environment.
2. Add the jar files for MySQL connection for Java.
3. Configure web.xml. Check for servlet-api.jar.
4. Create an HTML file and a DemoJDBC servlet.
5. Create a DBConnection class to initiate a JDBC connection.
6. Create config.properties file to store JDBC credentials.
7. Build, publish, and start the project.
8. Run the project.
9. Push the code to your GitHub repository.

Set up a JDBC Environment

Duration: 25 min.

Problem Statement:

Set up a JDBC environment.

Unassisted Practice: Guidelines

Steps to set up a JDBC environment:

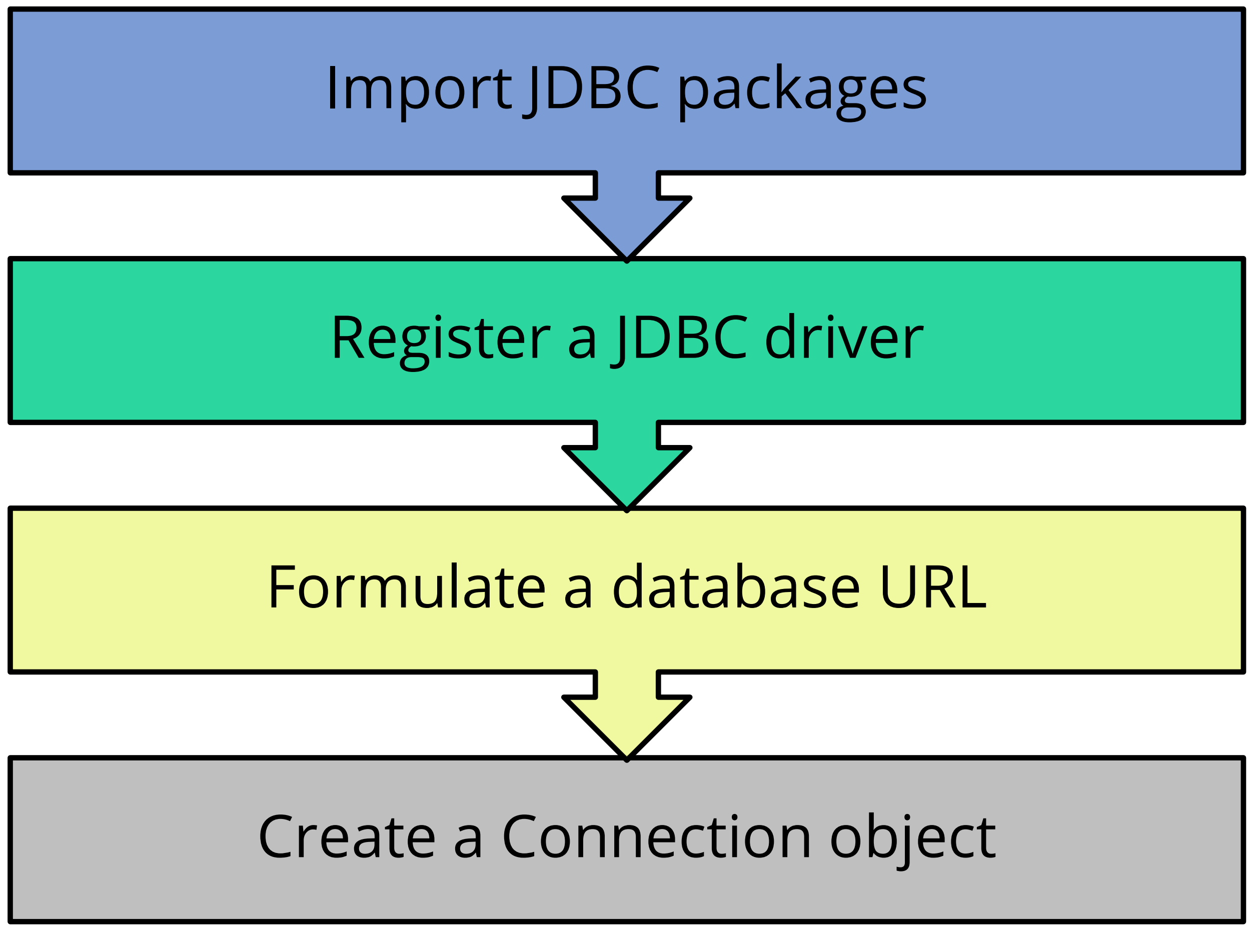
1. Create a dynamic web project to set up a JDBC environment.
2. Configure JDBC to connect to an existing MySQL database.
3. Create a servlet that creates a JDBC connection to the database and closes it.
4. Build, publish, and start the project.
5. Run the project.
6. Push the code to your GitHub repository.

JDBC Connections, Statement, and ResultSet

JDBC Connection

Connection to a relational database is represented by a JDBC connection. Connection is established using an object of Connection interface.

Steps to establish a JDBC connection:



JDBC Connection

A Connection object is instantiated using the DriverManager.getConnection method. The various overloaded methods of getConnection are listed below:

- getConnection (String urlString)
- getConnection (String urlString, Properties properties)
- getConnection (String urlString, String username, String password)

JDBC Statement

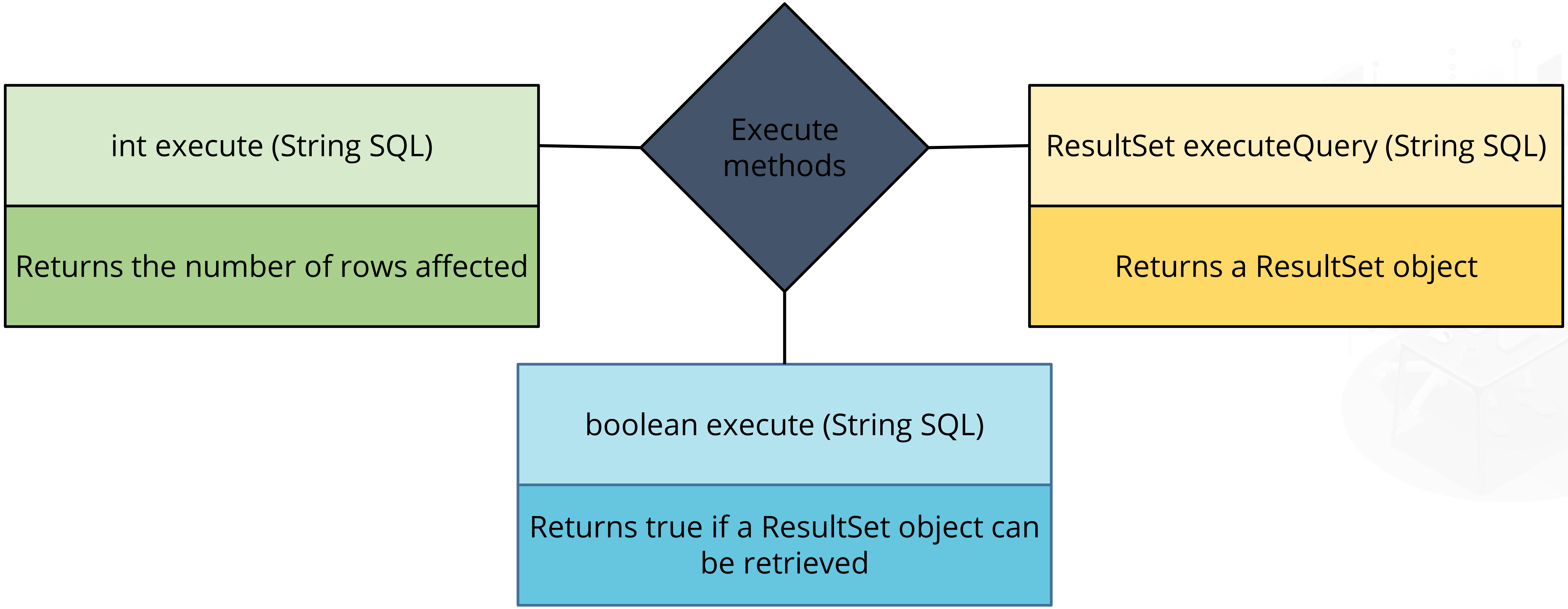
A JDBC Statement is used to execute queries. The methods in the Statement, CallableStatement, and PreparedStatement interfaces are used to execute queries against the database.

Statement Interfaces:

Interfaces	Description	Parameters
Statement	Used for general-purpose database access and executing static SQL statements at run-time	Parameters not accepted
CallableStatement	Used to access stored procedures in the database	Input parameters accepted at run-time
PreparedStatement	Used to execute SQL statements repeatedly	Input parameters accepted at run-time

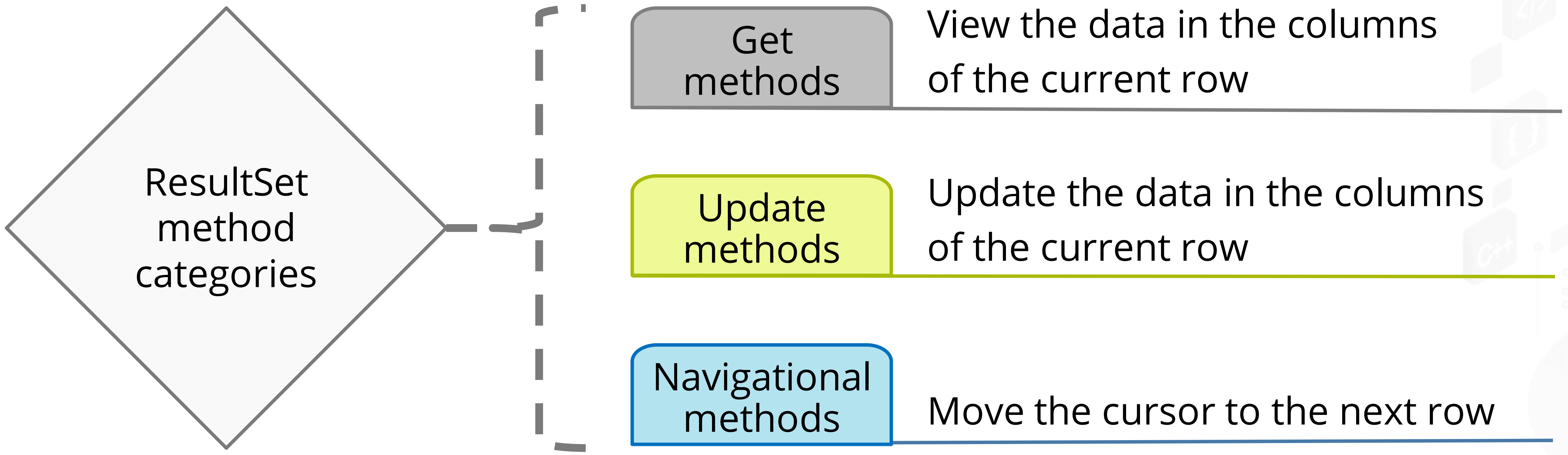
JDBC Statement

The Statement object uses the execute method to execute the SQL query.


Execute methods
int execute (String SQL)
Returns the number of rows affected
ResultSet executeQuery (String SQL)
Returns a ResultSet object
boolean execute (String SQL)
Returns true if a ResultSet object can be retrieved

JDBC ResultSet

Data read by SQL statements are returned in a result set. Result set of a database query is represented by the ResultSet interface.


ResultSet method categories
Get methods
View the data in the columns of the current row
Update methods
Update the data in the columns of the current row
Navigational methods
Move the cursor to the next row

ResultSet Concurrency

The concurrency of a ResultSet object controls the level of update functionality.

Concurrency	Description
CONCUR_READ_ONLY	The ResultSet object cannot be updated
CONCUR_UPDATABLE	The ResultSet object can be updated

By default, a ResultSet will be CONCUR_READ_ONLY.

Concurrency is not supported by all the JDBC drivers.

ResultSet Types

Based on cursor manipulation and concurrent changes to the underlying data source, a ResultSet object can be classified into three types: TYPE_FORWARD_ONLY, TYPE_SCROLL_INSENSITIVE, and TYPE_SCROLL_SENSITIVE.

ResultSet Type	Description	Sensitivity
TYPE_FORWARD_ONLY	Default ResultSet type: In the result set, the cursor can only go forward	-
TYPE_SCROLL_INSENSITIVE	The cursor can scroll forward and backward	Not sensitive to database changes that occur after the creation of the result set
TYPE_SCROLL_SENSITIVE	The cursor can scroll forward and backward	Sensitive to database changes that occur after the creation of the result set

JDBC Connections, Statements, and ResultSet

Duration: 45 min.

Problem Statement:

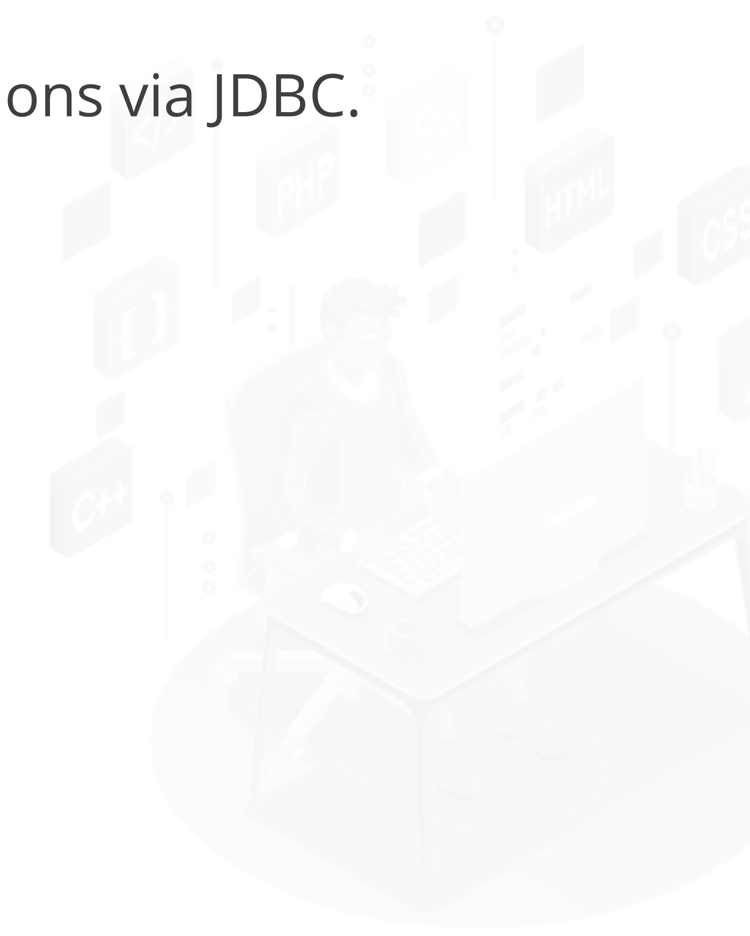
Demonstrate Connection, Statement, and ResultSet in JDBC.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate JDBC Connection, Statement, and ResultSet:

1. Create a database in MySQL and create a table in it.
2. Create a dynamic web project to connect to the database and perform data operations via JDBC.
3. Add the jar files for MySQL connection for Java.
4. Configure web.xml and check for servlet-api.jar.
5. Create an HTML file. Create a DBConnection class to initiate a JDBC connection.
6. Create config.properties file to store JDBC credentials.
7. Create a ProductDetails servlet to list data from the database.
8. Build, publish, and start the project. Run the project.
9. Push the code to your GitHub repository.



Stored Procedure and Exception Handling

Stored Procedure

A stored procedure is a prepared SQL code that can be saved and reused. It can contain more than one SQL statement.

A stored procedure is:

- A JDBC routine on the database side
- Defined in Java class and is stored in the database

Stored Procedure Types

Depending on the transactions in which the stored procedures are invoked, they are classified as non-nested connections and nested connections.

Nested connection	Uses the **same** transaction and connection of the parent SQL
Non-nested connection	Uses a **new** connection and a **different** transaction than that of the parent SQL

Steps to Call and Execute a Stored Procedure

A stored procedure is executed using a CallableStatement object.

1. Prepare a callable statement
2. Register the output parameters (if any)
3. Set the input parameters (if any)
4. Execute the CallableStatement and retrieve the result sets

Stored Procedure: Exception Handling

Exception handling in a stored procedure is database specific. An SQLException is thrown when an exception escapes the stored procedure.

SQLException Methods:

Method	Description
getErrorCode()	Returns the error number
getMessage()	Returns the error message
getSQLState()	Returns XOPEN SQLState string
getNextException	Returns the next Exception object in the exception chain
printStackTrace(PrintStream s)	Prints *this* throwable and its backtrace to the specified print stream
printStackTrace(PrintWriter w)	Prints *this* throwable and its backtrace to the specified print writer
printStackTrace( )	Prints the throwable and its backtrace or the current exception to a standard error stream

Stored Procedures and Exception Handling

Duration: 40 min.

Problem Statement:

Demonstrate stored procedures and exception handling in JDBC.

Assisted Practice: Guidelines

Steps to demonstrate stored procedures and exception handling:

1. Create a database in MySQL and create a table and a stored procedure.
2. Create a dynamic web project to connect to the database and execute a stored procedure.
3. Add the jar files for MySQL connection for Java.
4. Configure web.xml and check for servlet-api.jar.
5. Create an HTML file. Create a DBConnection class to initiate a JDBC connection.
6. Create config.properties file to store JDBC credentials.
7. Create a ProductDetails servlet to add a product using a stored procedure.
8. Build, publish, and start the project. Run the project.
9. Push the code to your GitHub repository.

Create, Select, and Drop a Database

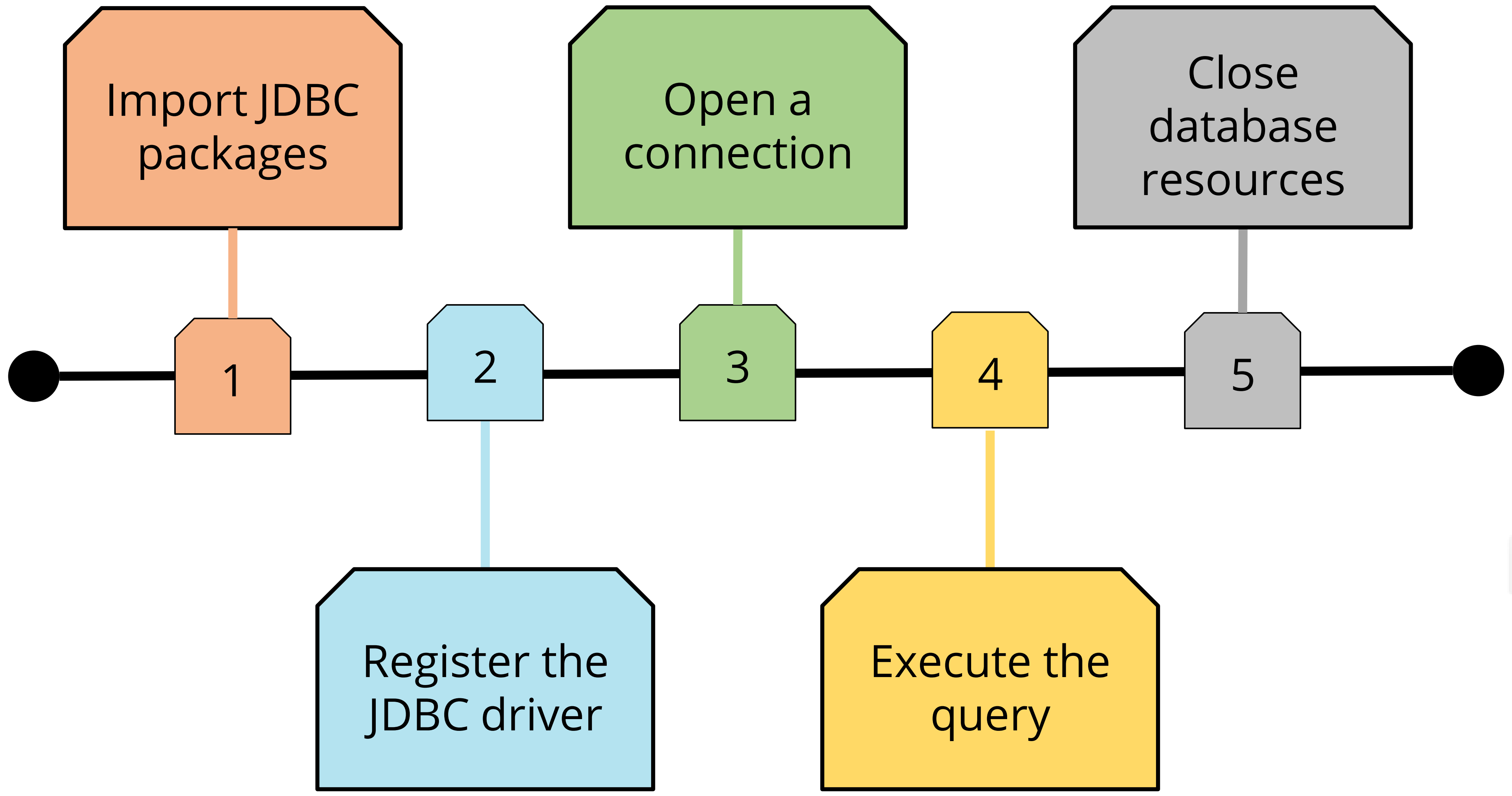
Create, Select, and Drop a Database

Before creating, selecting, and updating a database using JDBC, check if:

- The database is up and running
- The admin privilege is granted to create a new database in the given schema

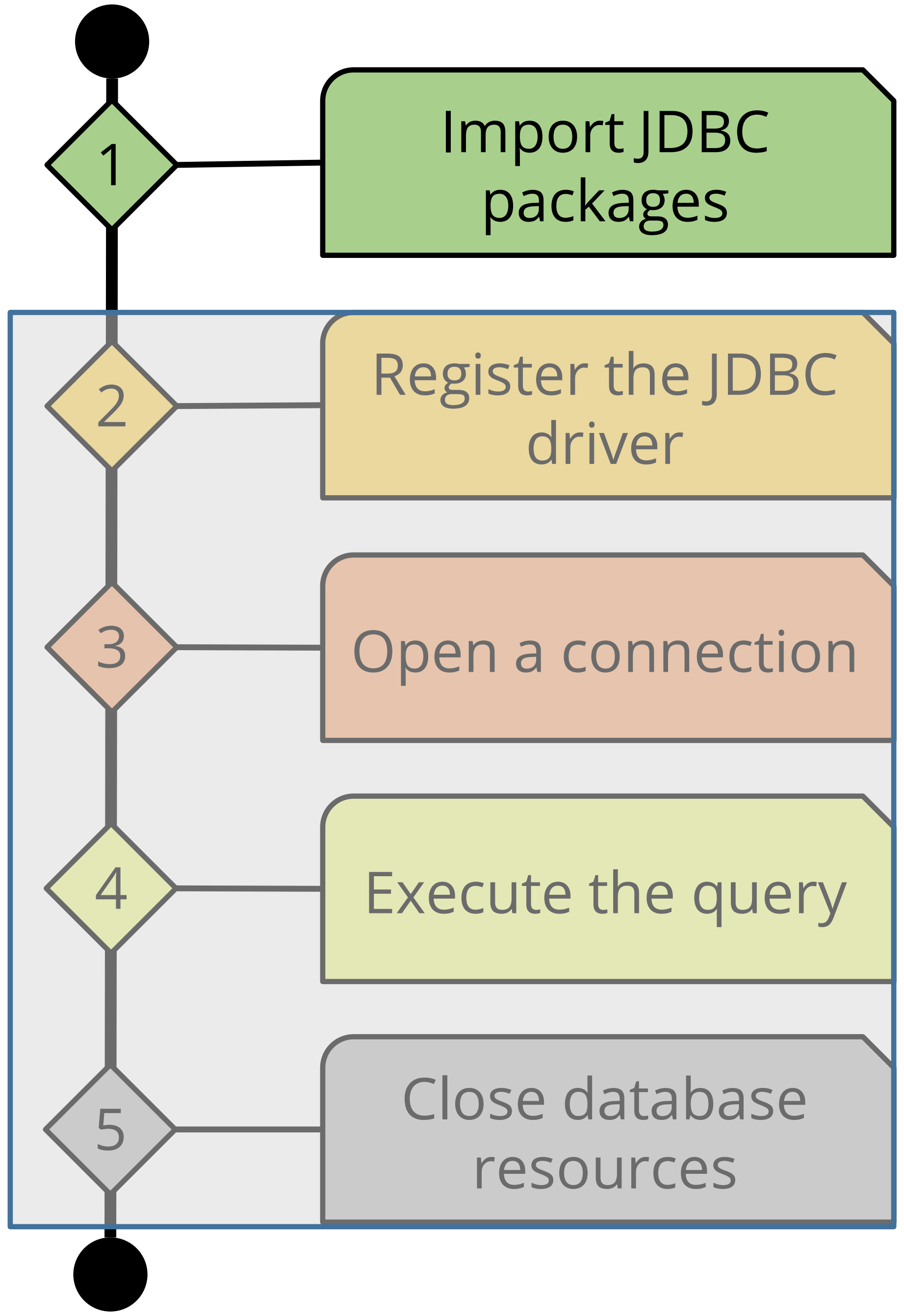
A JDBC Statement object is used to create, select, and drop a database

Steps to Create a Database



There is no need to provide a database name while preparing a database URL.

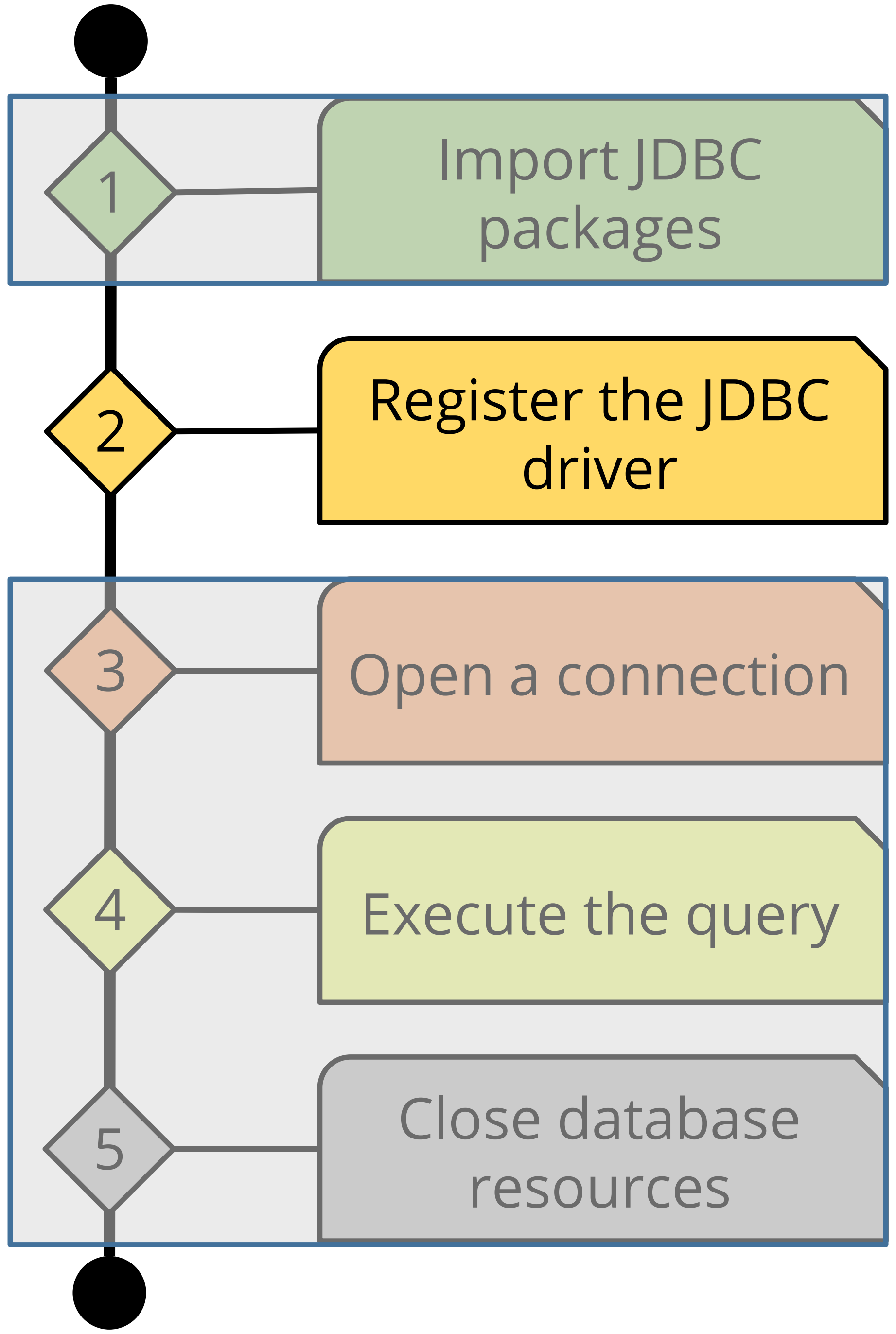
Create a Database



Sample Code

```
import java.sql.*;
// JDBC driver name
static final String  JDBC_DriverName =
"com.mysql.jdbc.Driver";
// URL String
static final String database_URL =
"jdbc:mysql://localhost";
// Database credentials
static final String  userName =
“NameOfTheUser";
static final String password =
"password";
```

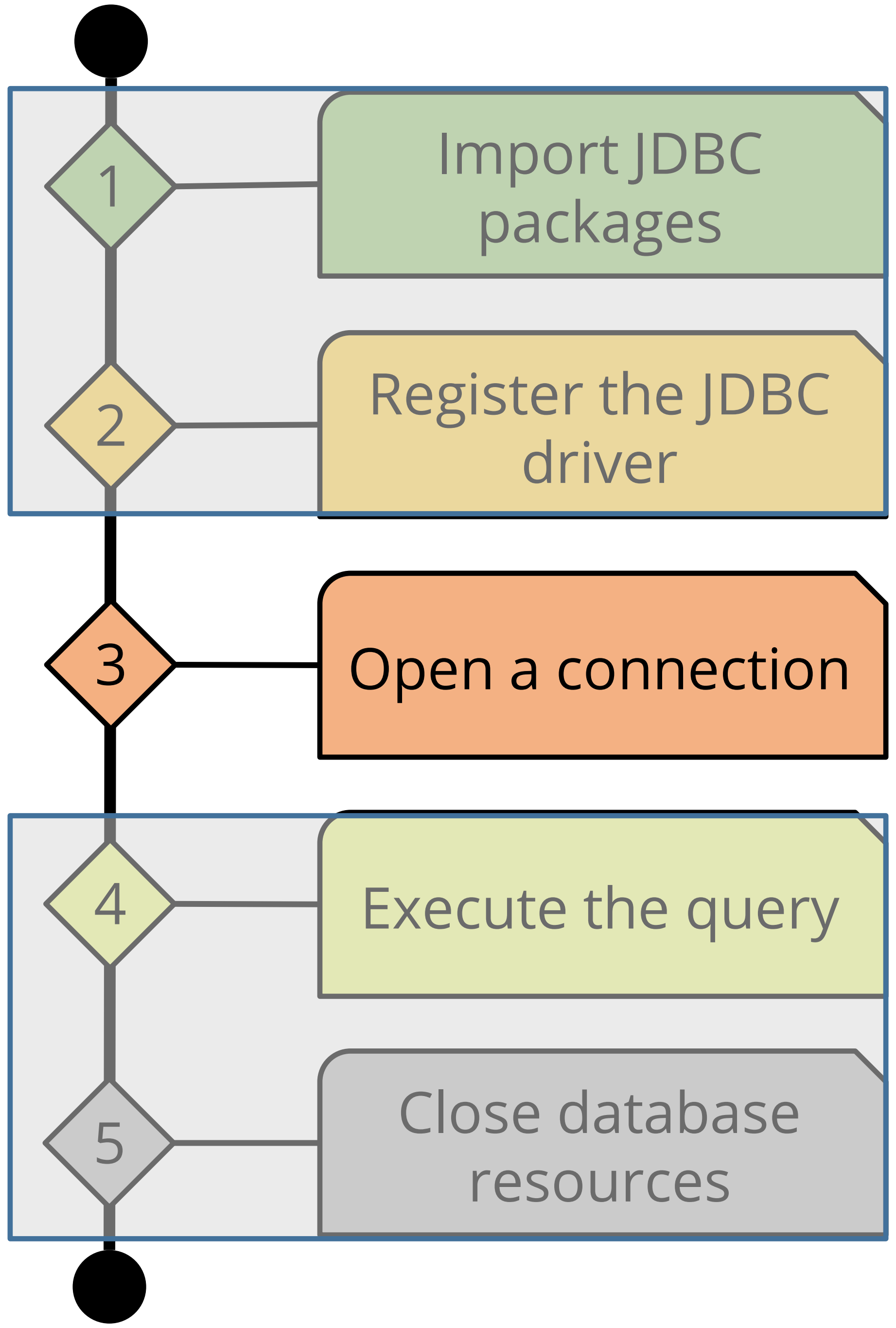
Create a Database



Sample Code

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

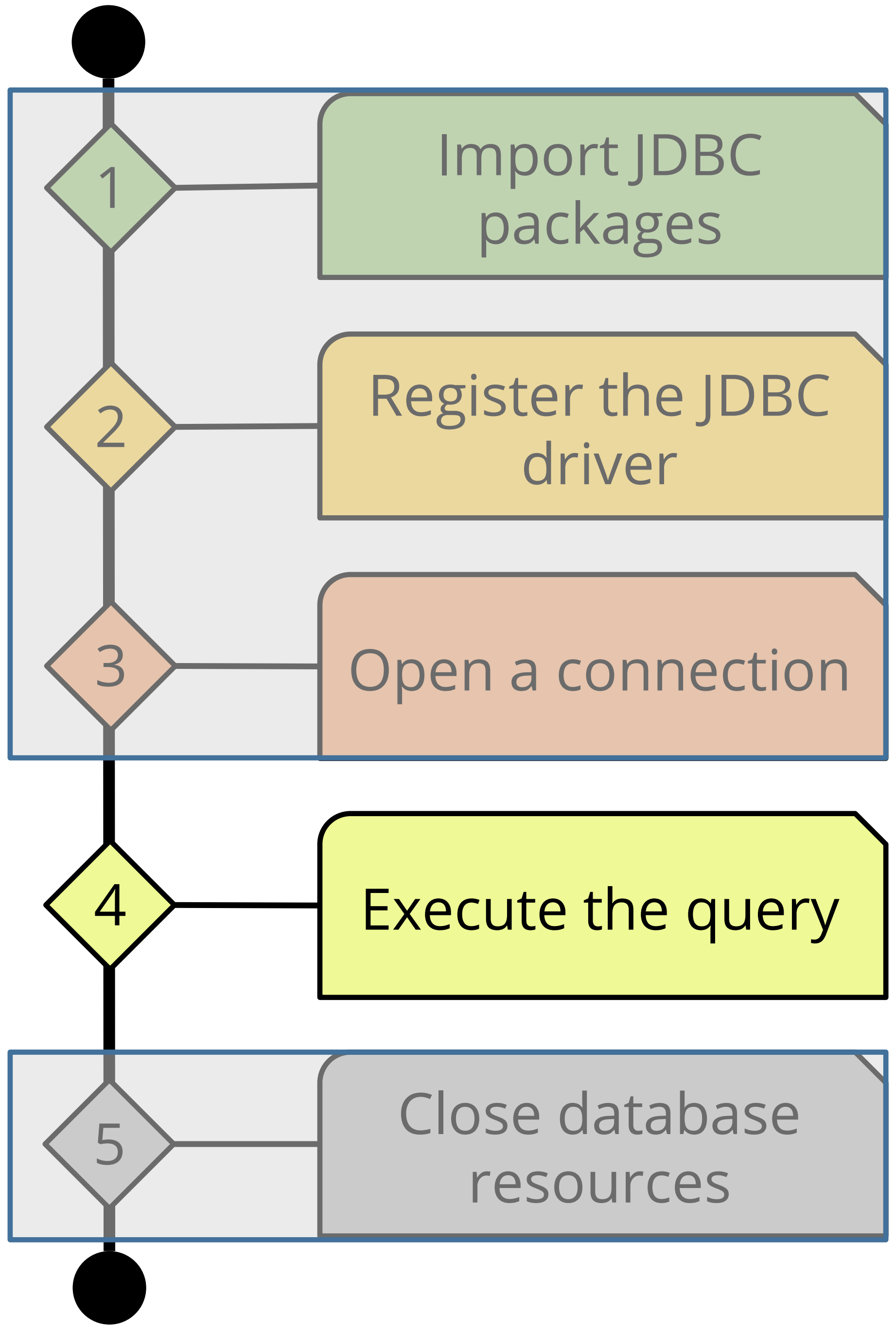
Create a Database



Sample Code

```
Connection connect =
DriverManager.getConnection(database_URL,
userName, password);
```

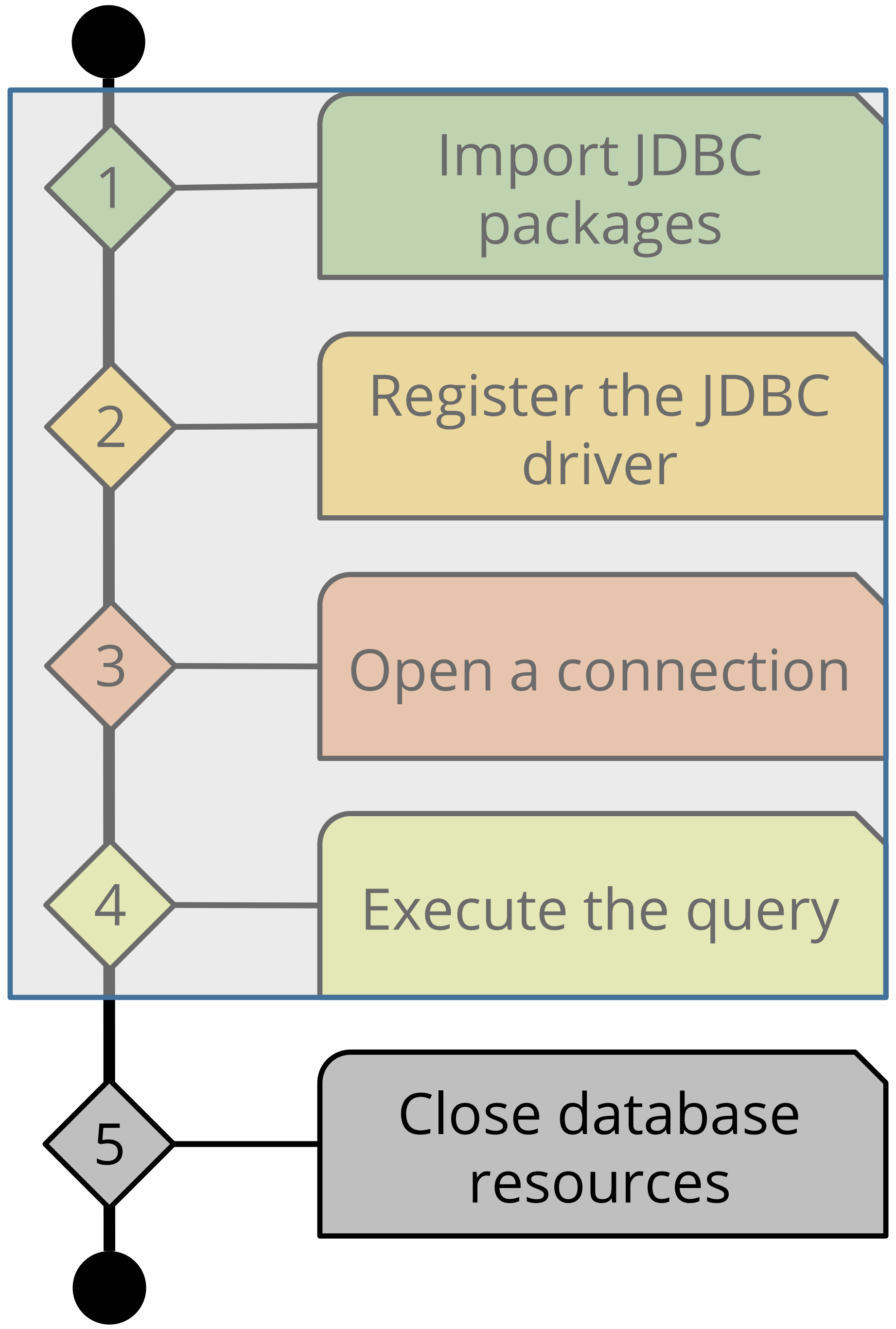
Create a Database



Sample Code

```
Statement stmt = connect.createStatement();
String sqlString = "CREATE DATABASE SAMPLE";
stmt.executeUpdate(sqlString);
```

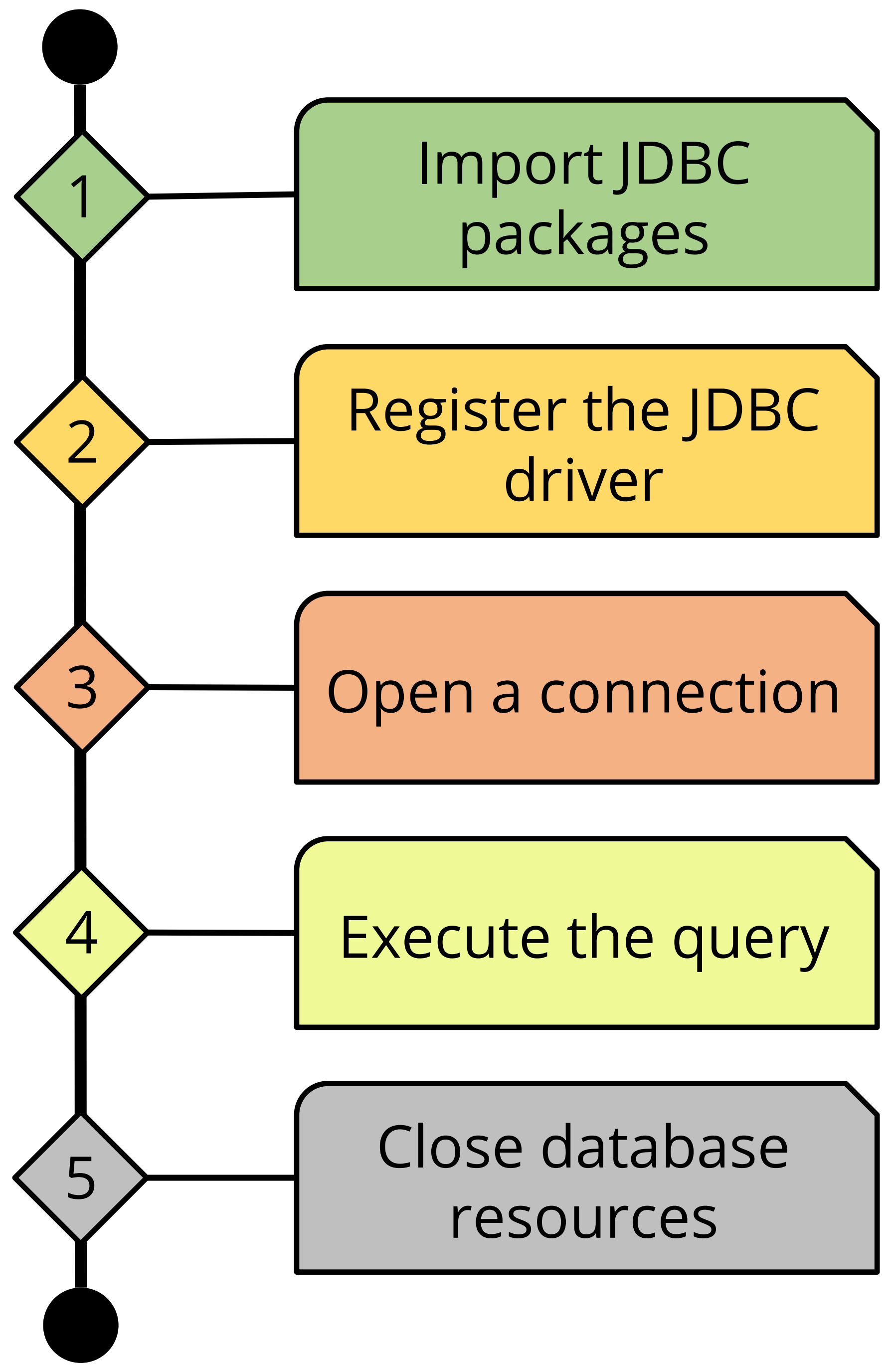
Create a Database



Sample Code

```
stmt.close(); connect.close();
```

Steps to Select a Database

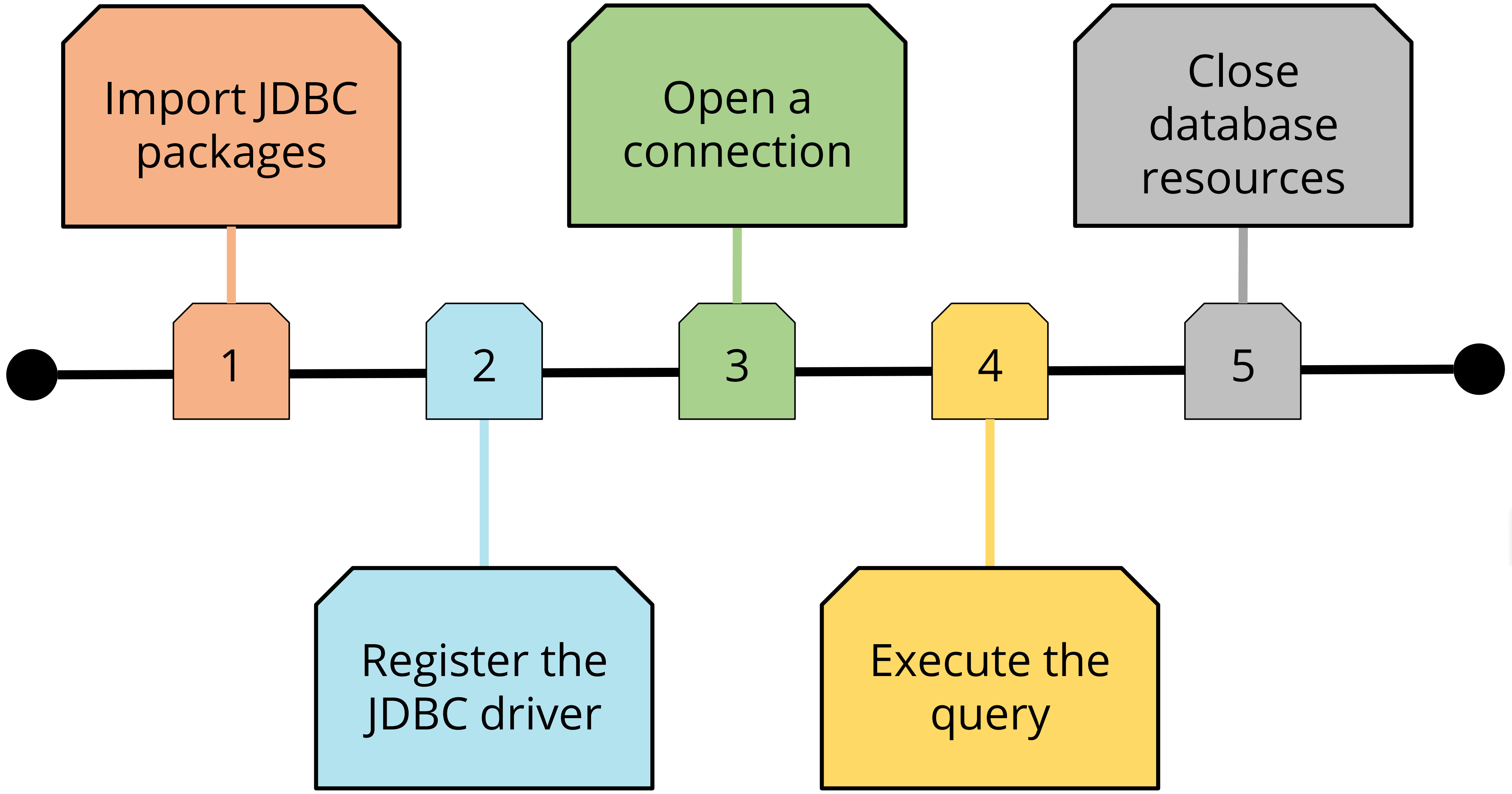


The name of the database to be selected must be mentioned in the database URL

Sample Code

```
// URL String
static final String database_URL
="jdbc:mysql://localhost/databaseName";
```

Steps to Drop a Database



Steps to Drop a Database

The name of the database to be dropped must be mentioned in the database URL.

Sample Code

```
// Database to be dropped mentioned in the URL String

   static final String database_URL
     ="jdbc:mysql://localhost/databaseName";

// Drop the database
   Statement stmt = connect.createStatement();
   String sqlString = “DROP DATABASE SAMPLE”;
   stmt.executeUpdate(sqlString);
```

Create, Select, and Drop a Database

Duration: 45 min.

Problem Statement:

Demonstrate how to create, select, and drop a database in JDBC.

Assisted Practice: Guidelines

Steps to create, select, and drop a database

1. Create a dynamic web project to use JDBC to create, select, and drop a database.
2. Add the jar files for MySQL connection for Java.
3. Configure web.xml and check for servlet-api.jar.
4. Create an HTML file. Create a DBConnection class to initiate a JDBC connection.
5. Create config.properties file to store JDBC credentials.
6. Create a DBOperations servlet to create, select, and drop a database.
7. Build, publish, and start the project. Run the project.
8. Push the code to your GitHub repository.

Create, Select, and Drop a Database

Duration: 45 min.

Problem Statement:

Demonstrate how to create, select, and drop a database in JDBC.

UNASSISTED PRACTICE

Unassisted Practice: Guidelines

Steps to create, select, and drop a database

1. Create a dynamic web project to use JDBC to create, select, and drop a database.
2. Add the jar files for MySQL connection for Java.
3. Configure web.xml and check for servlet-api.jar.
4. Create an HTML file. Create a DBConnection class to initiate a JDBC connection.
5. Create config.properties file to store JDBC credentials.
6. Create a DBOperations servlet to create, select, and drop a database.
7. Build, publish, and start the project. Run the project.
8. Push the code to your GitHub repository.

Insertion, Updation, and Deletion of Database Records

Insert, Update, and Delete Records

Insert a record in the table, update an existing record, or delete a record in the table by following the steps below:

- Establish a connection to the database
- Create a Statement object to perform the query
- Execute INSERT, UPDATE. or DELETE statement
- Close the database connection

Insert a Record

An **INSERT** statement is used to insert a record in a database table.

Example

```
//Connection and statement objects
      Connection connect = null;
      Statement stmt = null

// Database to be connected to
   static final String database_URL =
   "jdbc:mysql://localhost/databaseName";

// Insert a record into SampleTable
   stmt = connect.createStatement();
   String sqlString = "INSERT INTO employeeRecords " +
"VALUES
  (100,'firstNameTest','lastNameTest')";
   stmt.executeUpdate(sqlString);
```

Update a Record

An **UPDATE** statement is used to update an existing record in the database table.

Example

```
//Connection and statement objects
      Connection connect = null;
      Statement stmt = null

// Database to be connected to
   static final String database_URL =
   "jdbc:mysql://localhost/databaseName";

// Insert a record into SampleTable
   stmt = connect.createStatement();
   String sqlString = "UPDATE employeeRecords " + "SET
   lastName = Rao WHERE id in (101, 203)";
   stmt.executeUpdate(sqlString);
```

Delete a Record

A **DELETE** statement is used to delete an existing record from the database table.

Example

```
//Connection and statement objects
      Connection connect = null;
      Statement stmt = null

// Database to be connected to
   static final String database_URL =
   "jdbc:mysql://localhost/databaseName";

// Insert a record into SampleTable
   stmt = connect.createStatement();
   String sqlString = “DELETE FROM employeeRecords“
   + “WHERE id=101";
   stmt.executeUpdate(sqlString);
```

Insertion, Updation, and Deletion of Database Records

Duration: 45 min.

Problem Statement:

Demonstrate database record handling using JDBC.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to insert, update, and delete records from a database:

1. Create a dynamic web project to use JDBC.
2. Create a database in MySQL and create a table in it.
3. Add the jar files for MySQL connection for Java.
4. Configure web.xml and check for servlet-api.jar.
5. Create an HTML file. Create a DBConnection class to initiate a JDBC connection.
6. Create config.properties file to store JDBC credentials.
7. Create a ProductDetails servlet to add, update, and delete records from the table.
8. Build, publish, and start the project. Run the project.
9. Push the code to your GitHub repository.

Transaction Management

Transaction Management: JDBC

- Transaction represents a single unit of work.
- The ACID (Atomicity, Consistency, Isolation and Durability) properties describe transaction management.
  - Atomicity means that every transaction is successful, or nothing is successful.
  - Consistency ensures that the database is brought from one consistent state to another consistent state.
  - Isolation ensures that each transaction is isolated from the other transaction.
  - Durability means that once a transaction has been committed, it will remain so, even in the event of errors or power loss.

Transaction Management: JDBC

Statement: It is used for general-purpose access to your database. It is useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters.

```
Statement stmt = null;
try
{
   stmt = conn.createStatement( );
   . . . }
catch (SQLException e) {
   . . . }
finally {
   . . .}
```

Transaction Management: JDBC

PreparedStatement: The PreparedStatement interface extends the Statement interface and provides you added functionality. It has a few advantages over a generic Statement object.

The prepareStatement() method of Connection interface is used to return the object of PreparedStatement.

Syntax:

```
public PreparedStatement prepareStatement(String query)throws SQLExce
ption{}
```

Transaction Management: JDBC

The important methods of PreparedStatement interface are given below:

Method	Description
public void setInt(int paramIndex, int value)	sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used to create, drop, insert, update, delete, etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

Transaction Management: JDBC

Example: PreparedStatement:

```
PreparedStatement pstmt = null;
try
{
   String SQL = "Update Employees SET age = ? WHERE id = ?";
   pstmt = conn.prepareStatement(SQL);
   . . . }
catch (SQLException e) {
   . . . }
finally {
   . . . }
```

Transaction Management: JDBC

Transaction management in JDBC using Statement

```
import java.sql.*;
class FetchRecords
{
public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection  con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:x
e","system","oracle");
con.setAutoCommit(false);
Statement stmt=con.createStatement();
stmt.executeUpdate("insert into user420 values(190,'abhi',40000)");
stmt.executeUpdate("insert into user420 values(191,'umesh',50000)");
con.commit();
con.close();
}}
```

Transaction Management: JDBC

Example: Transaction Management in JDBC using PreparedStatement

```
import java.sql.*;
import java.io.*;
class TM
{
public static void main(String args[]){
try{
  class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
con.setAutoCommit(false);
PreparedStatement ps=con.prepareStatement("insert into user420 values(?,?,?)");
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
while(true){
System.out.println("enter id");
String s1=br.readLine();
int id=Integer.parseInt(s1);
System.out.println("enter name");
String name=br.readLine();
System.out.println("enter salary");
String s3=br.readLine();
```

Transaction Management: JDBC

Transaction Management in JDBC using PreparedStatement

```
int salary=Integer.parseInt(s3);
ps.setInt(1,id);
ps.setString(2,name);
ps.setInt(3,salary);
ps.executeUpdate();
System.out.println("commit/rollback");
String answer=br.readLine();
if(answer.equals("commit")){
con.commit();  }
if(answer.equals("rollback")){
con.rollback();  }
System.out.println("Want to add more records y/n");
String ans=br.readLine();
if(ans.equals("n")){
break;
} }
con.commit();
System.out.println("record successfully saved");
con.close();//before closing connection commit() is called
}catch(Exception e){System.out.println(e);}  }}
```

Transaction Management: JDBC

CallableStatement interface: It is used to call the stored procedures and functions.

An example for getting the instance of CallableStatement:

```
CallableStatement stmt=con.prepareCall("{call myprocedure(?,?)}");
```

An example for calling the function using JDBC:

First, create a function in the database:

```
create or replace function sum4
(n1 in number,n2 in number)
return number
is
temp number(8);
begin
temp :=n1+n2;
return temp;
end;
/
```

Transaction Management: JDBC

Once you have created a function in the database, call that function:

```
import java.sql.*;
public class FuncSum {
public static void main(String[] args) throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
CallableStatement stmt=con.prepareCall("{?= call sum4(?,?)}");
stmt.setInt(2,10);
stmt.setInt(3,43);
stmt.registerOutParameter(1,Types.INTEGER);
stmt.execute();
System.out.println(stmt.getInt(1));
}
}
```

Transaction Management

Duration: 30 min.

Problem Statement:

Demonstrate Transaction Management in JDBC.

Assisted Practice: Guidelines

Steps to perform Transaction Management:

1. Use Auto-Commit mode for Transaction Management.
2. Disable setAutoCommit() and demonstrate Transaction Management.
3. Push the code to your GitHub repositories.

Key Takeaways

- JDBC stands for Java Database Connectivity. It is a Java API used to connect to database and execute the query with the database
- The connection between a Java application and database is a connection. It is used to get the objects of Statement and DatabaseMetaData
- The statement interface is used to provide methods to execute queries over database
- The Resultset object maintains a cursor pointing to a row of a table in the database
- ACID property describes the transaction management very well

Create Your Database Using JDBC

Duration: 45 min.

Problem Statement: Create a Java program to retrieve the product details of a particular product.

LESSON-END PROJECT

Before the Next Class

Course: Software Testing: Get a high paying Job In Technology

You should be able to:

- Explain software testing and software tester
- Describe the methodologies used in software testing
- List the differences between manual testing and automation testing
- Demonstrate what bugs are and how to find these bugs