Creating Your Database Using JDBC

You Already Know

Course(s):

SQL Training

Recap

- Group data using queries
  - What are databases?
  - Tables and relations

- Demonstrate SQL queries
  - What is querying?
  - Dml, ddl, and more
  - Common query tool

- Filter results using queries
  - Creating a filter condition
  - Applying multiple filter conditions

Recap

- Group data using queries
  - The group by clause

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

A Day in the Life of a Full Stack Developer

As Joe has good experience in database handling, a new project related to the database is assigned to him.

To make the user experience better, Joe has to 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:

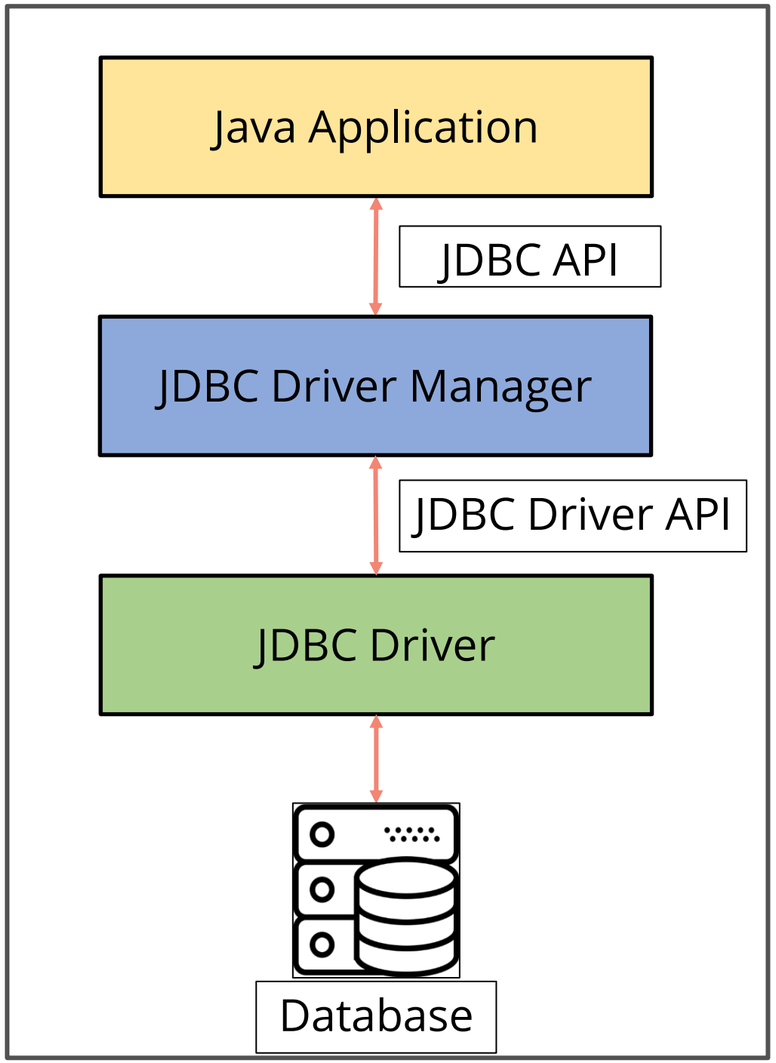
- Explain JDBC
- Illustrate a connection in JDBC
- Describe statement and its types
- Demonstrate resultset and its types
- Describe stored procedures
- Explain exception handling in stored procedures
- Describe database creation, selection, and dropping
- Demonstrate insertion, updation, and deletion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

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.

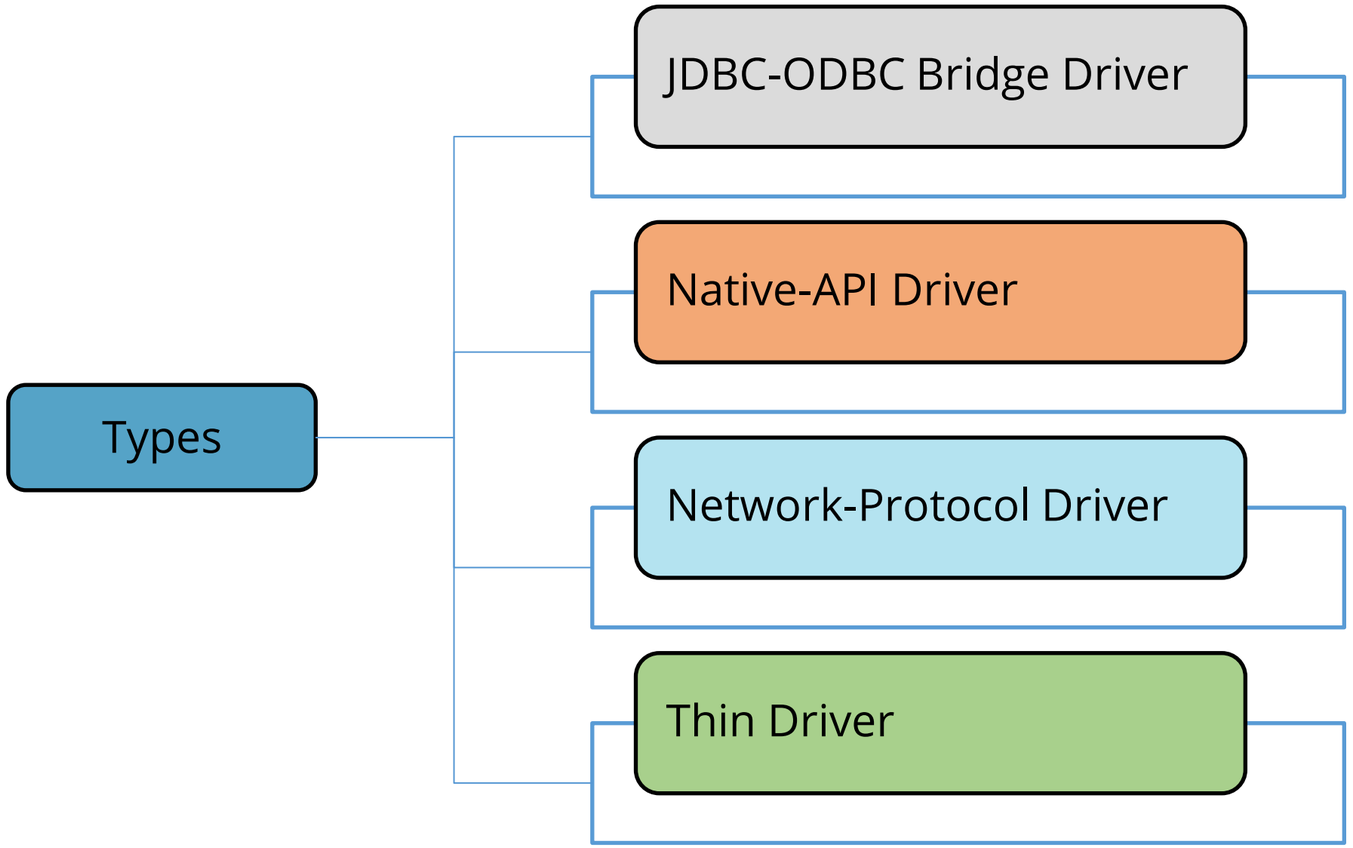
JDBC Architecture



Latest JDBC version:
JDBC 4.3

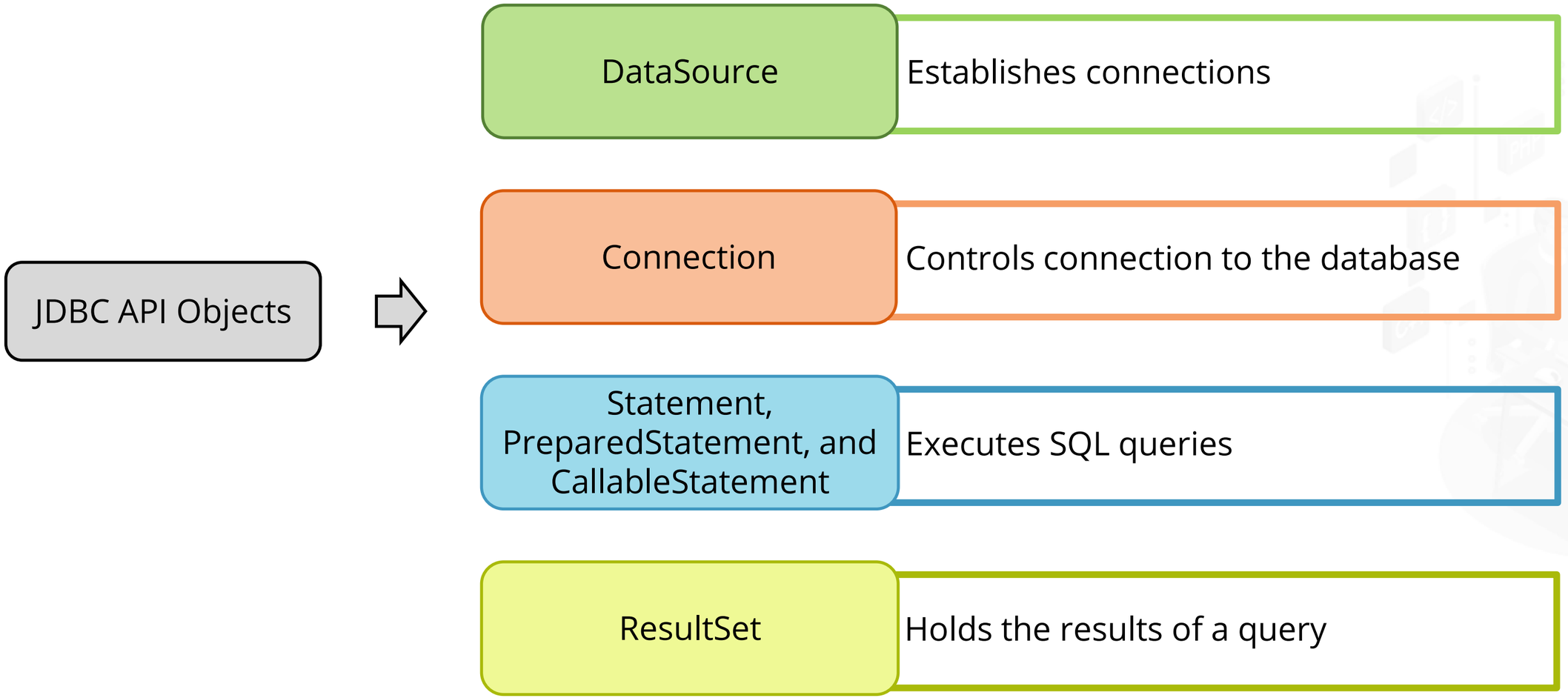
JDBC Driver and Its Types

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

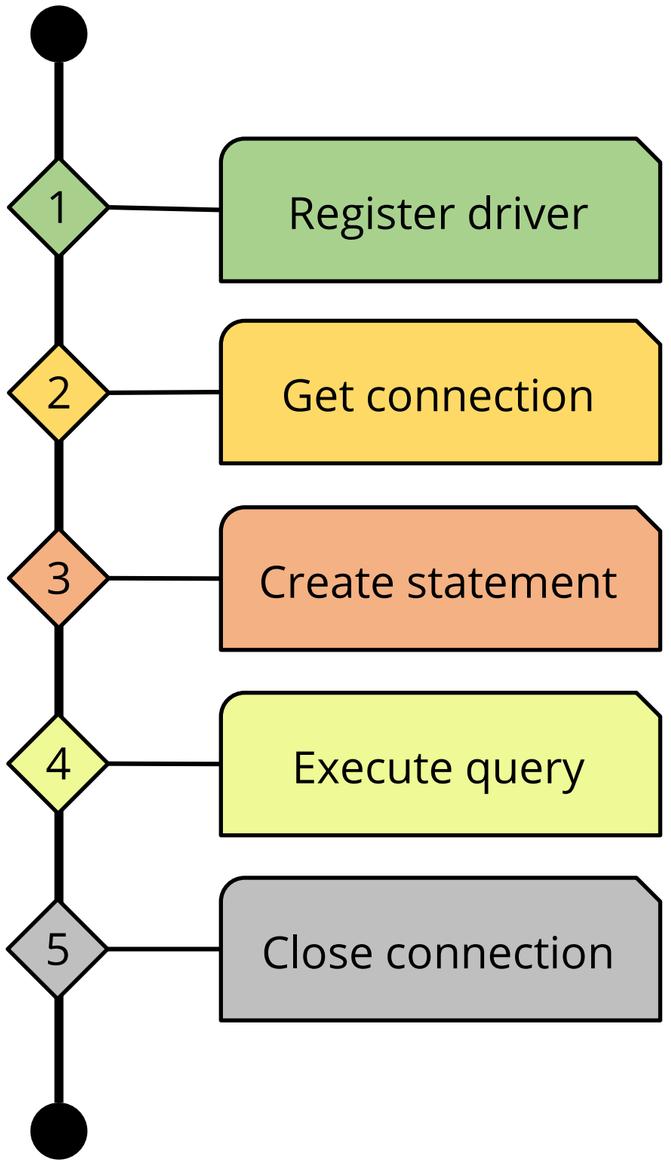


JDBC API

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



JDBC Connectivity



Set Up JDBC Environment

Duration: 25 min.

Problem Statement:

Demonstrate a project to set up JDBC environment.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to set up 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 and 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 JDBC Environment

Duration: 25 min.

Problem Statement:

Demonstrate a project to set up JDBC environment.

UNASSISTED PRACTICE

Unassisted Practice: Guidelines

Steps to set up JDBC environment:

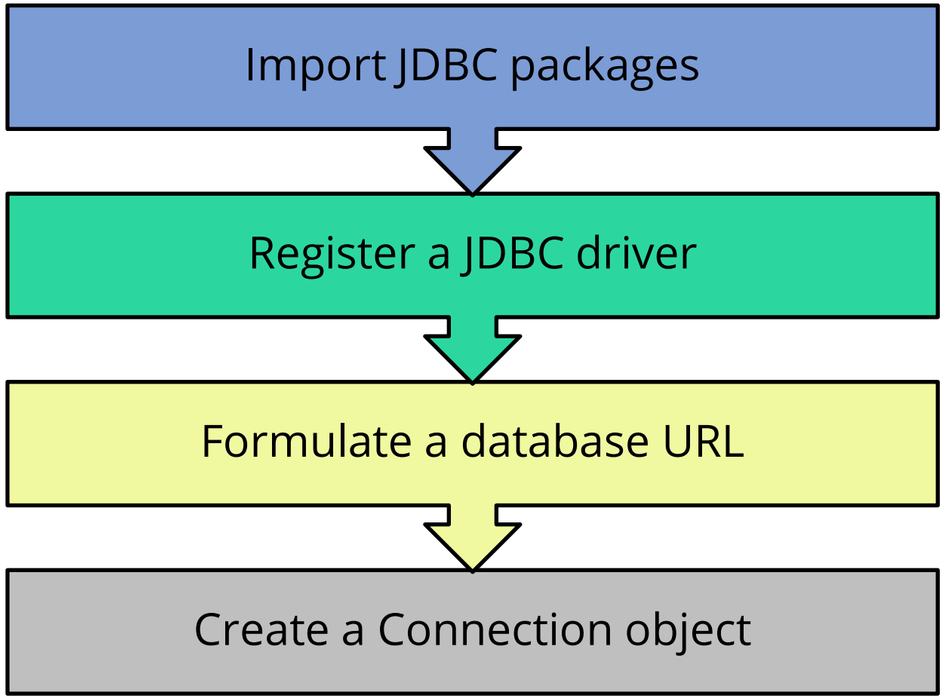
1. Create a dynamic web project to set up a JDBC environment.
2. Configure JDBC to connect to an existing MySQL database.
3. Create a servlet that creates a JDBC connection to the database and closes it.
4. Build, publish, and start the project.
5. Run the project.
6. Push the code to your GitHub repository.

JDBC Connections, Statement, and ResultSet

JDBC Connection

The connection to a relational database is represented by a JDBC connection. This 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

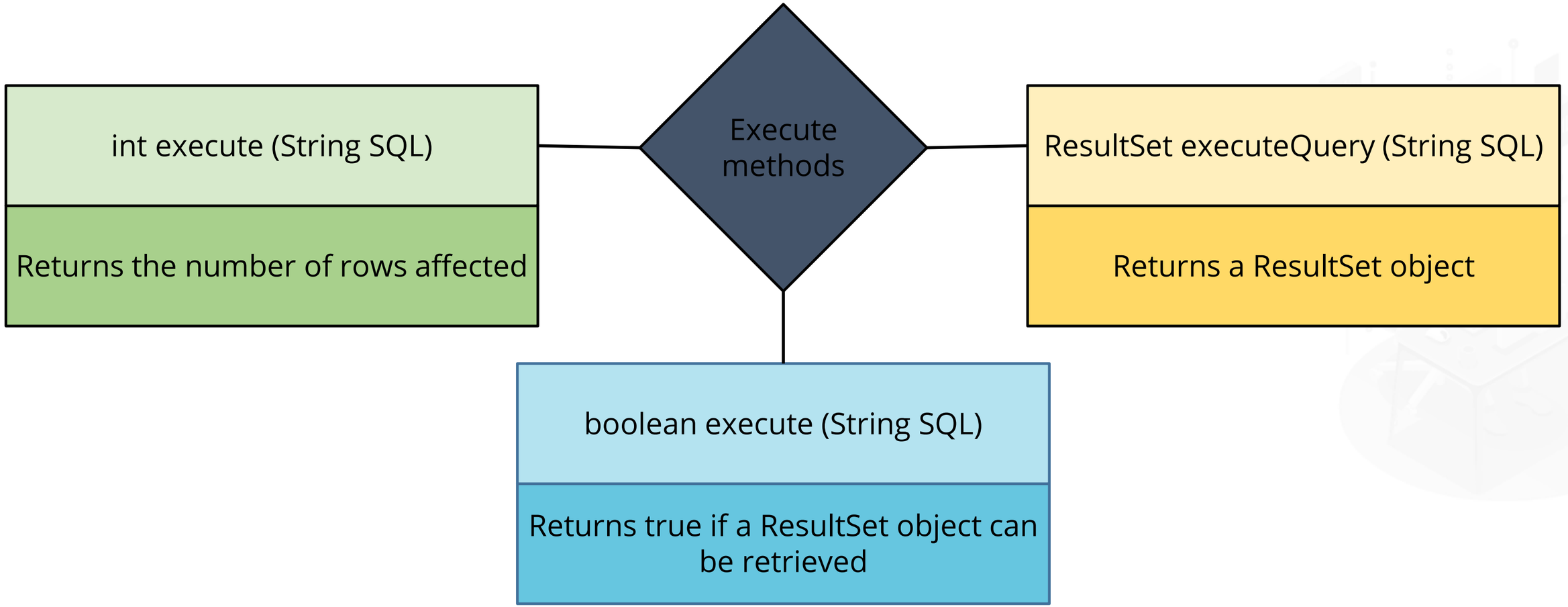
A JDBC Statement is used to execute queries. The methods in the Statement, CallableStatement, and PreparedStatement interfaces are used to execute queries against the database.

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

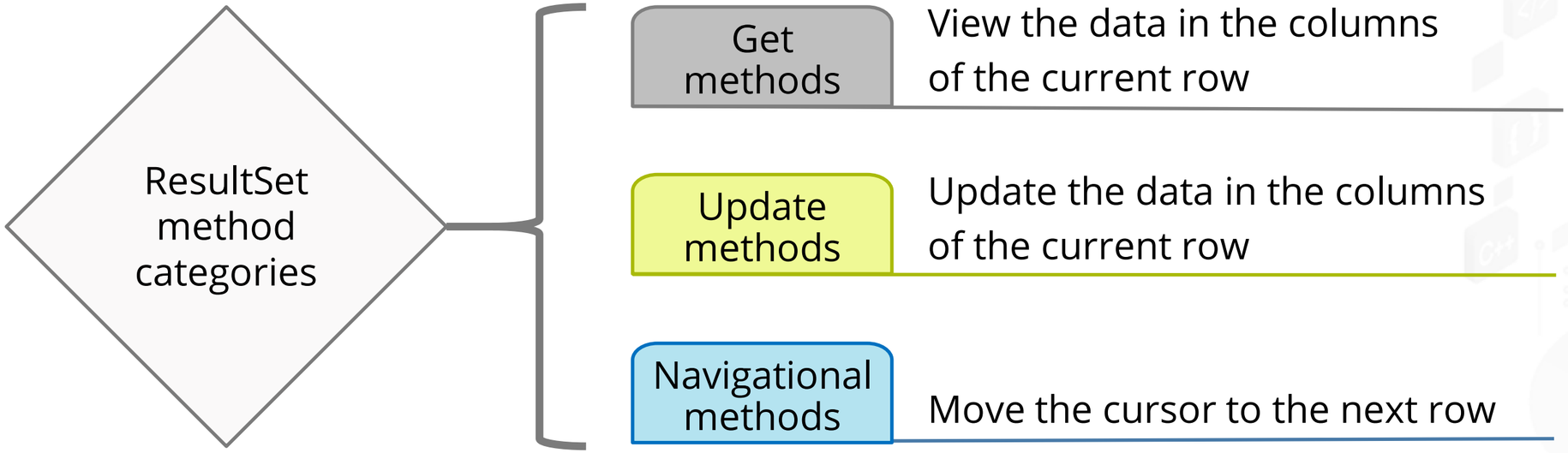
JDBC Statement

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



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.



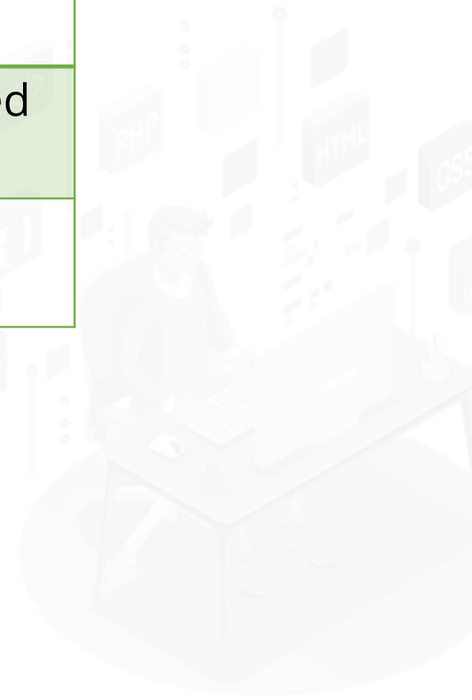
ResultSet Concurrency

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

Types of Stored Procedure

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.

Prepare a callable statement

Register the output parameters (if any)

Set the input parameters (if any)

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

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

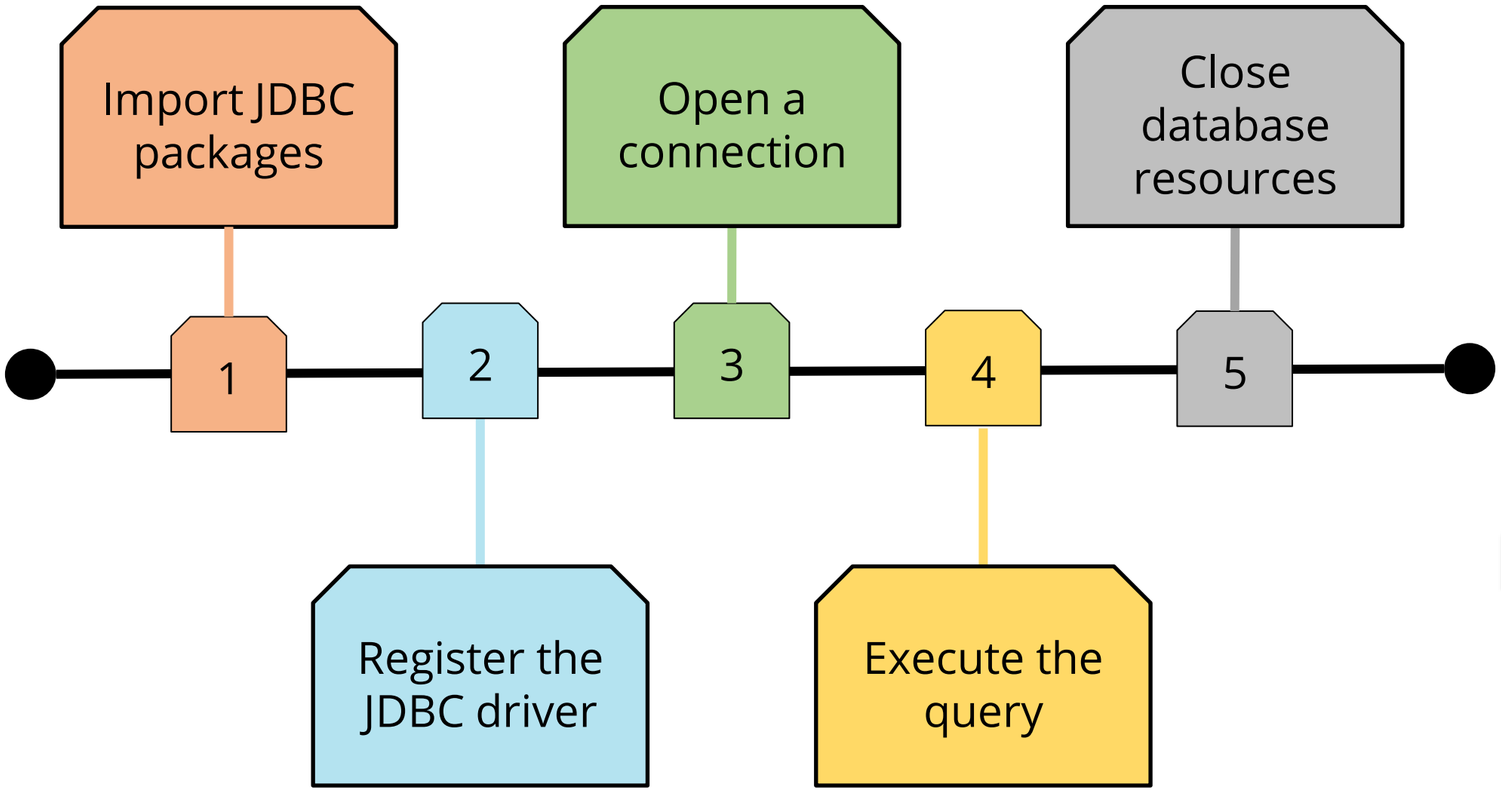
Create, Select, and Drop a Database

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

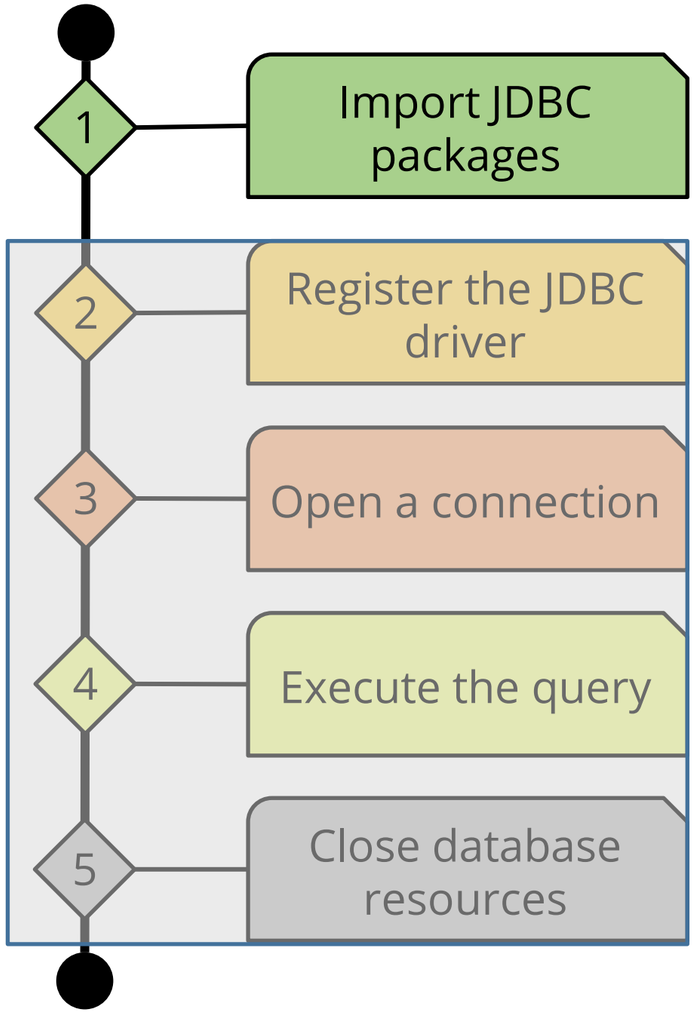
A JDBC Statement object is used to create, select, and drop a database

Steps to Create a Database



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

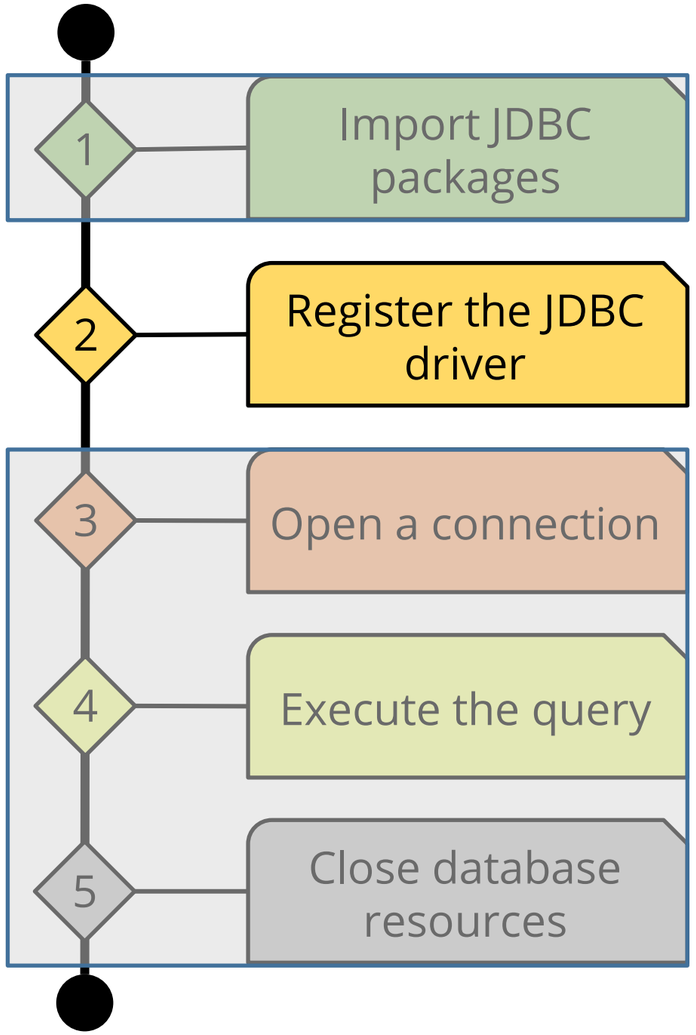
Create a Database



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";
```

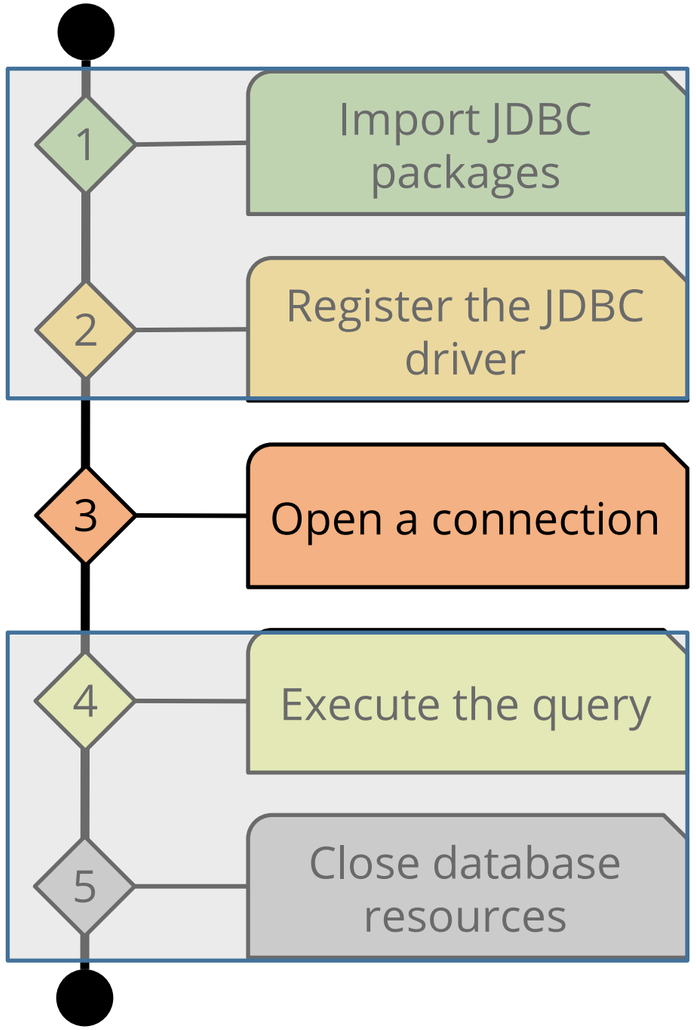
Create a Database



Sample Code

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

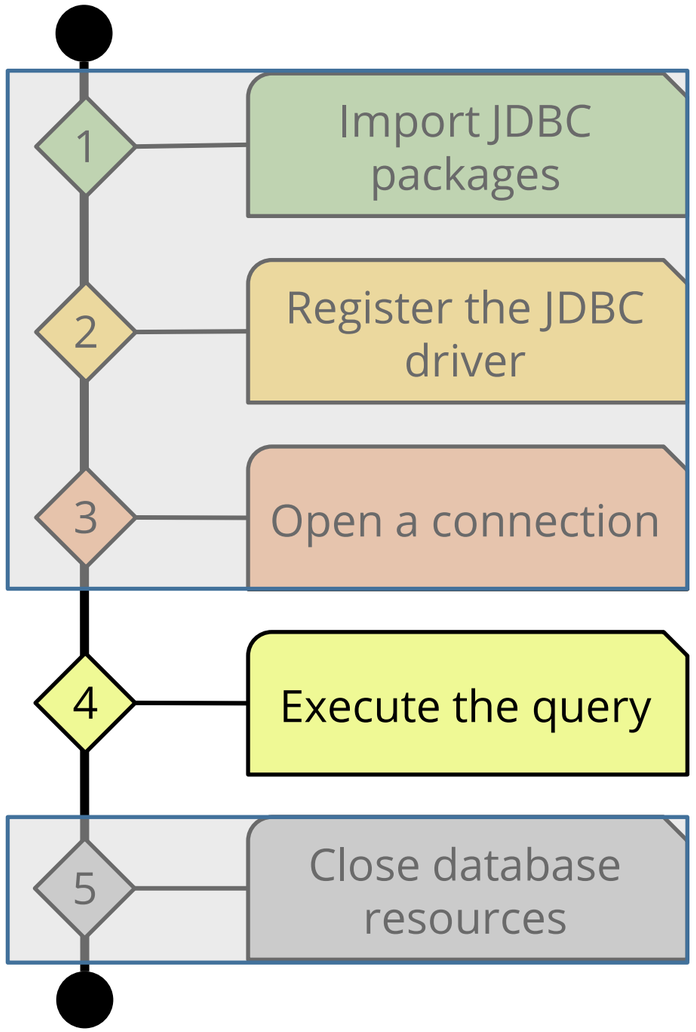
Create a Database



Sample Code

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

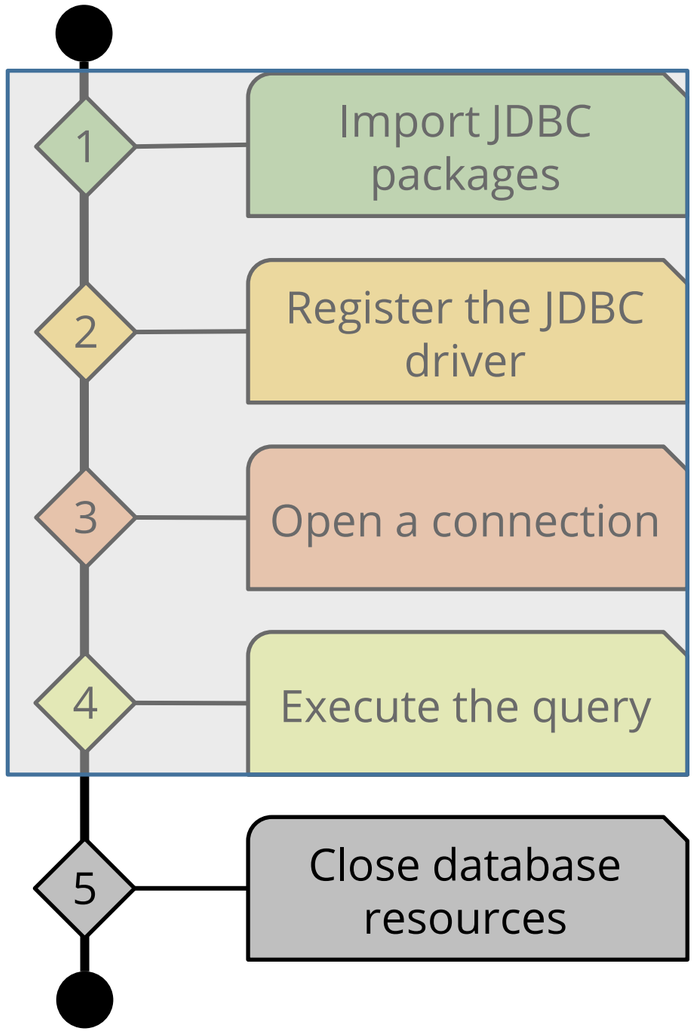
Create a Database



Sample Code

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

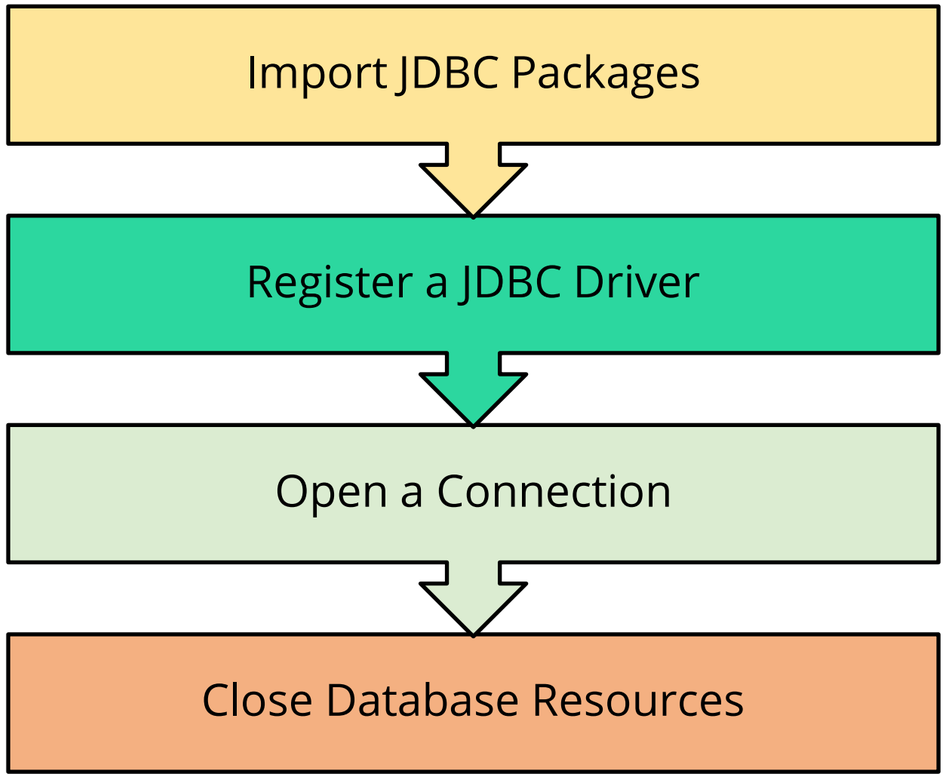
Create a Database



Sample Code

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

Steps to Select a Database

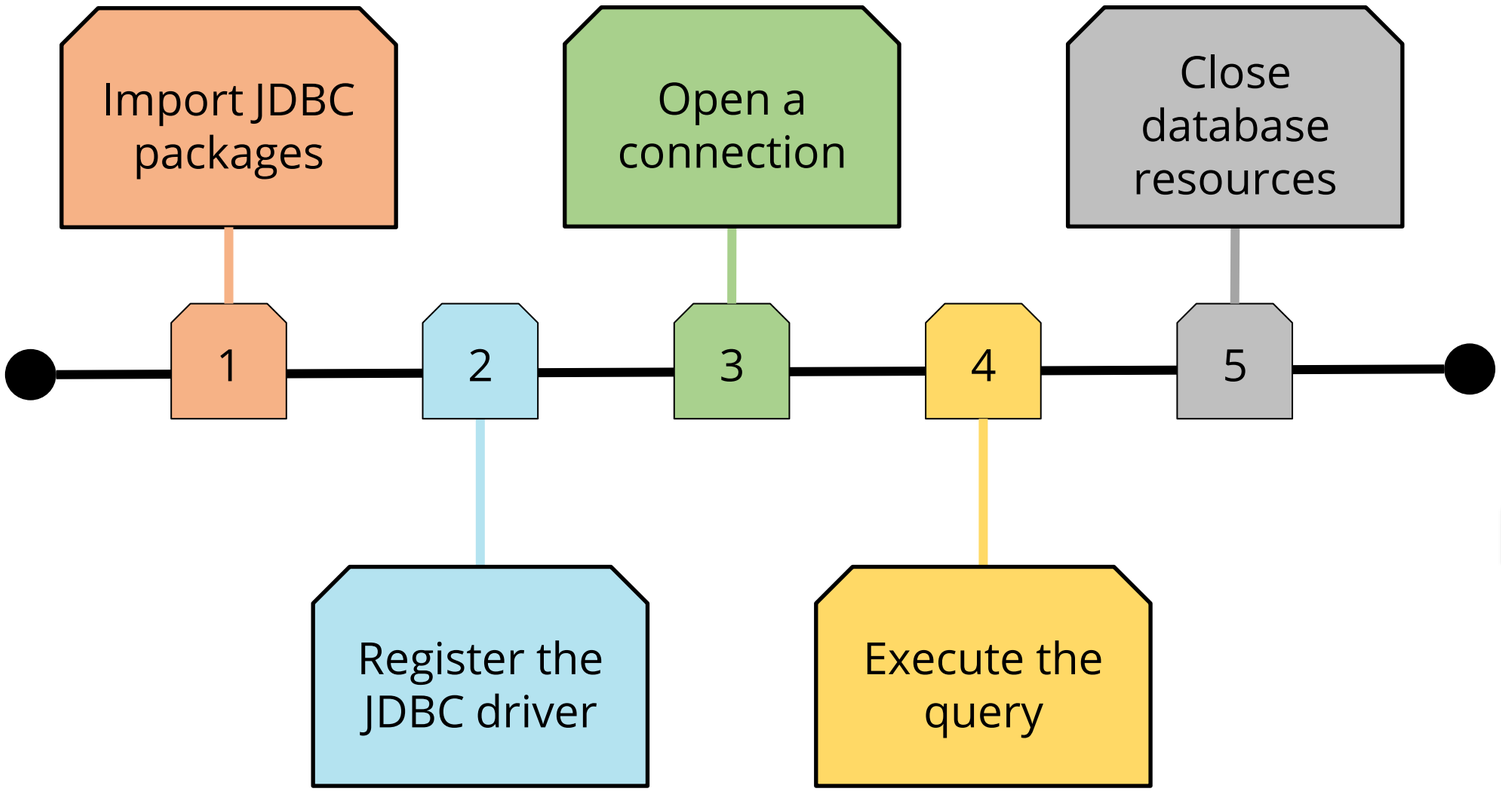


The name of the database to be selected must be mentioned in the database URL

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

Assisted Practice: Guidelines

Steps to demonstrate JDBC database actions:

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 DB Operations 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

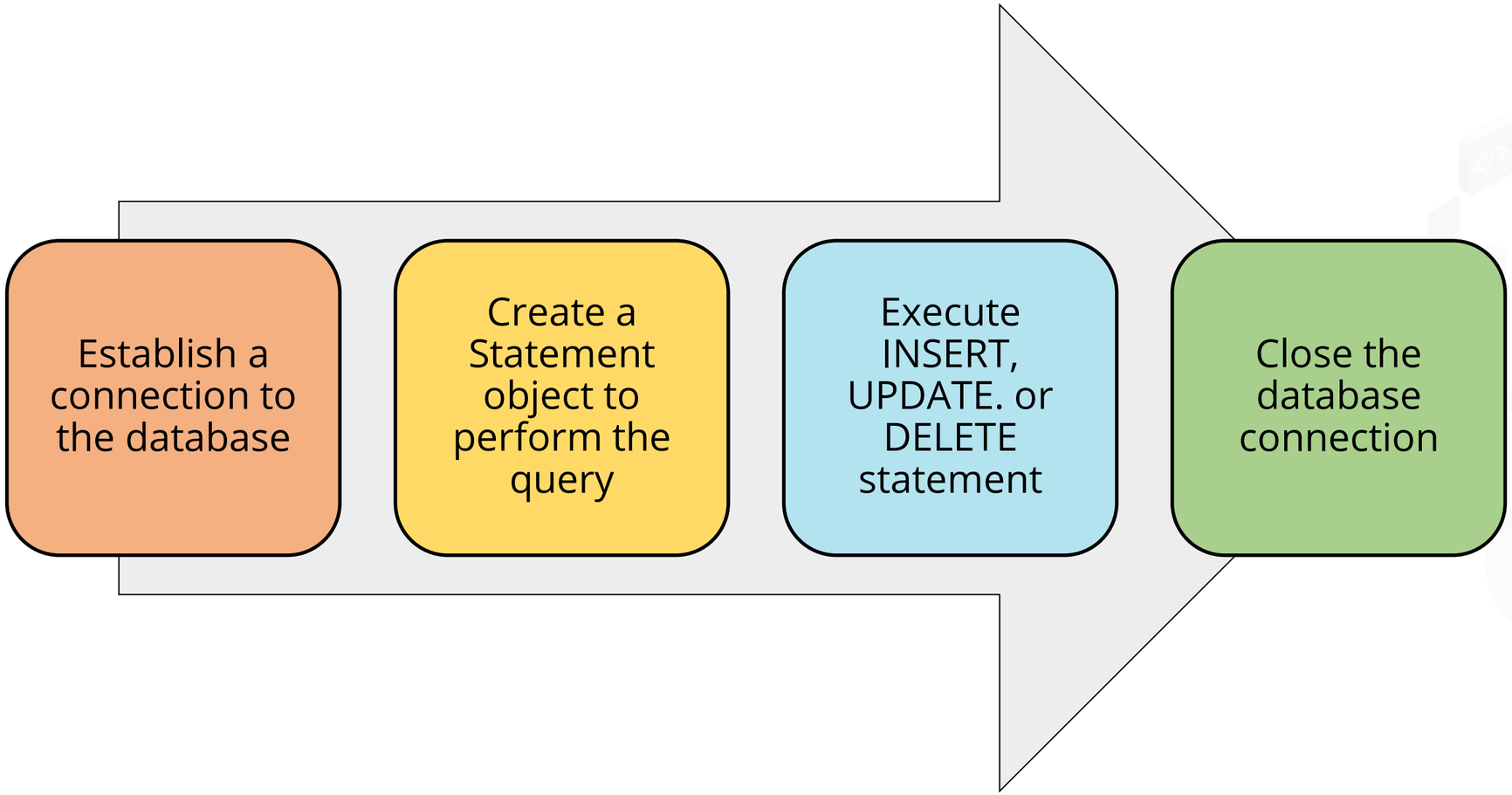
Steps to demonstrate JDBC database actions:

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 DB Operations 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:



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 demonstrate database record handling:

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.

Key Takeaways

- JDBC stands for Java Database Connectivity. It is a Java API used to connect to a database and execute the query with the database.
- 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.

Retrieving the Product Details Using the Product ID

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: Java Certification Training Course

You should be able to:

- Explain Hibernate
- Demonstrate CRUD operations
- Use Hibernate queries and relationships
- Demonstrate Hibernate mapping