Coding Bootcamp

©Simplilearn. All rights reserved.

simplilearn

JDBC

©Simplilearn. All rights reserved.

TECHNOLOGY

Getting Started with JDBC

©Simplilearn. All rights reserved.

Learning Objectives

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

- Apply JDBC setup and configuration to initiate and manage Java projects effectively
- Establish and manage JDBC connections with a MySQL database, and perform CRUD operations using JDBC
- Implement transaction management techniques and handle SQL exceptions using JDBC for secure and reliable database operations
- Execute batch processing commands in JDBC to enhance database performance and efficiency

©Simplilearn. All rights reserved.

Learning Objectives

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

- Utilize various types of JDBC drivers to facilitate database connectivity in Java applications
- Apply knowledge of JDBC architecture and key components to configure and optimize Java applications for database interactions
- Utilize savepoints in JDBC to effectively manage and rollback transactions during database operations
- Execute stored procedures using JDBC to streamline and enhance database tasks and operations

©Simplilearn. All rights reserved.

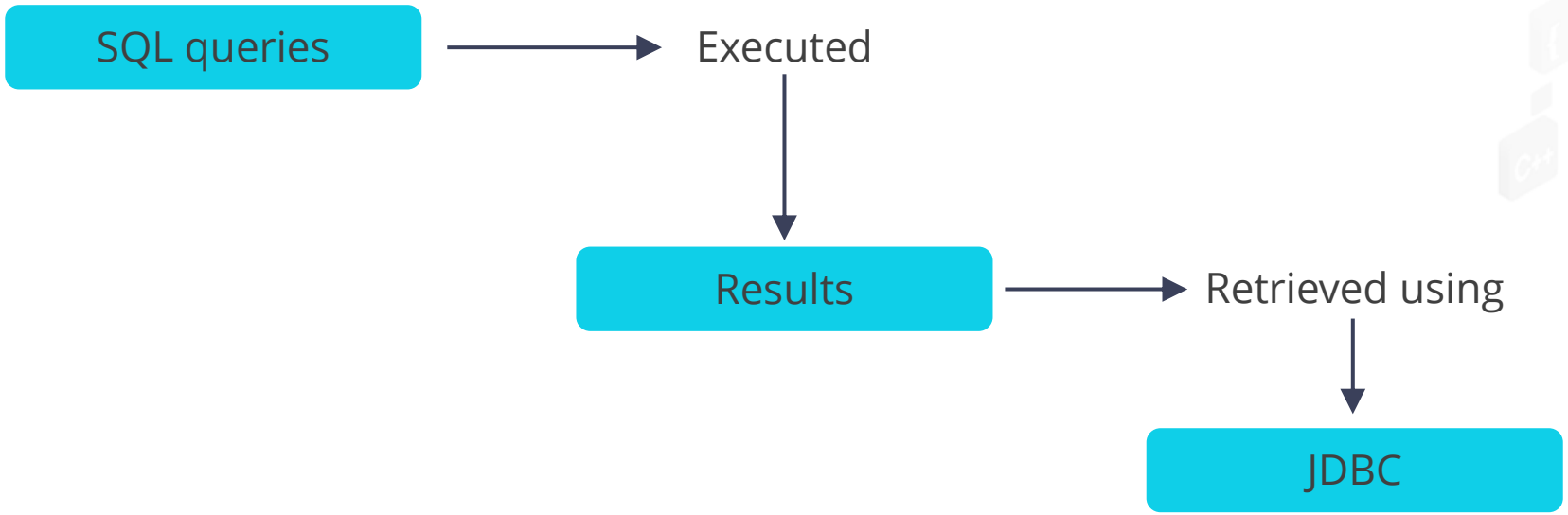
JDBC

©Simplilearn. All rights reserved.

JDBC

Java Database Connectivity (JDBC) is a platform-independent interface that connects Java applications to databases.

SQL queries are executed with the help of JDBC functions, and the results are stored in Java variables.



©Simplilearn. All rights reserved.

JDBC

JDBC is an API specifically built to enable Java applications to interact with a wide range of data sources.

Specifically

built for

©Simplilearn. All rights reserved.

Why Use JDBC?

Here are some key purposes of JDBC:

Database connectivity	• It provides a standard API for database interaction and abstracts the complexity involved in database communication.
SQL execution	• It supports batch processing of SQL statements for efficient bulk updates and queries.
Data access	• It offers structured access to database data through Java objects, reducing the complexity of data handling.

Why Use JDBC?

Database independence	• It reduces dependency on database-specific code, promoting application portability across different database platforms.
Transaction management	• It provides mechanisms to control and manage database transactions to ensure data integrity and consistency.

Where to Use JDBC?

JDBC is used in various industries, such as:

Web applications

- It allows CRUD operations in web apps.
- It enhances efficiency with pooled database connections.

Enterprise applications

- It ensures data integrity with complex transactions.
- It seamlessly integrates with Java EE for robust solutions.

Desktop applications

- It links database actions with GUI elements.
- It connects to both local and remote databases.

Where to Use JDBC?

JDBC is used in various industries, such as:

Batch processing

- It manages large data transactions efficiently.
- It executes database-related scheduled tasks.

Data integration

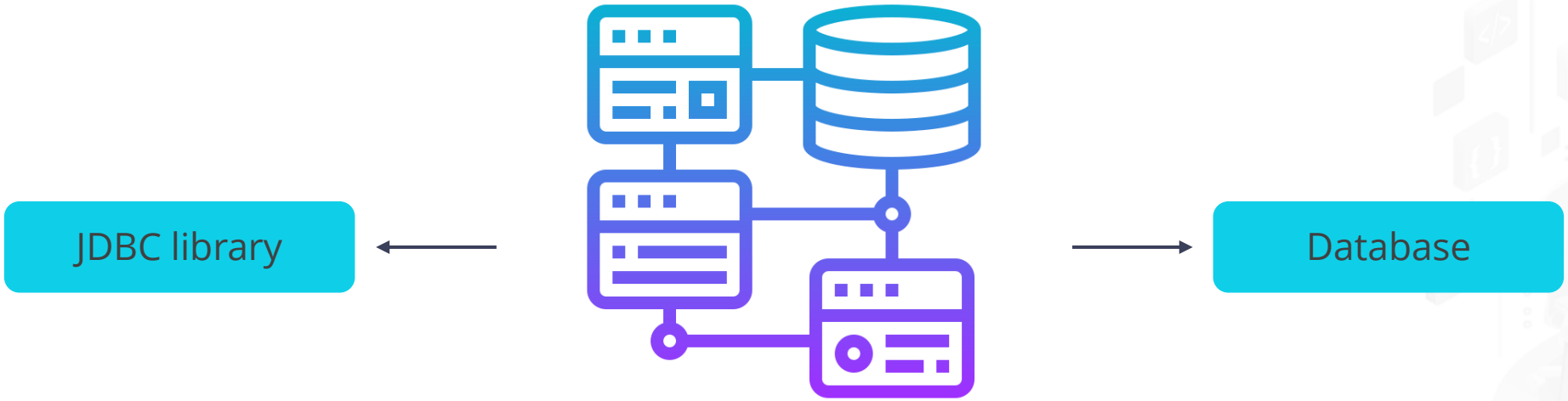
- It facilitates communication across different databases.
- It allows switching between databases without code changes.

Reporting and analytics

- It retrieves large datasets for analysis.
- It generates real-time and data-driven reports.

JDBC Library

It includes APIs for common database-related tasks.



©Simplilearn. All rights reserved.

JDBC Library

It helps to:

Make the connection to the database

Create the SQL or MySQL statements

Execute SQL or MYSQL queries in the database

View and modify the resulting records

©Simplilearn. All rights reserved.

JDBC Library

It offers a comprehensive collection of interfaces and libraries that facilitate seamless connectivity to an underlying database.

©Simplilearn. All rights reserved.

JDBC Library

It can be used to write different types of executables, such as:

Java Applications

Java Applets

Java Servlets

Java Server Pages (JSP)

Enterprise Java Beans (EJB)

JDBC allows Java programs to contain database-independent code.

©Simplilearn. All rights reserved.

SQL

©Simplilearn. All rights reserved.

SQL

SQL stands for Structured Query Language.

It is a programming language for accessing and manipulating databases.

©Simplilearn. All rights reserved.

SQL

SQL became a standard of:

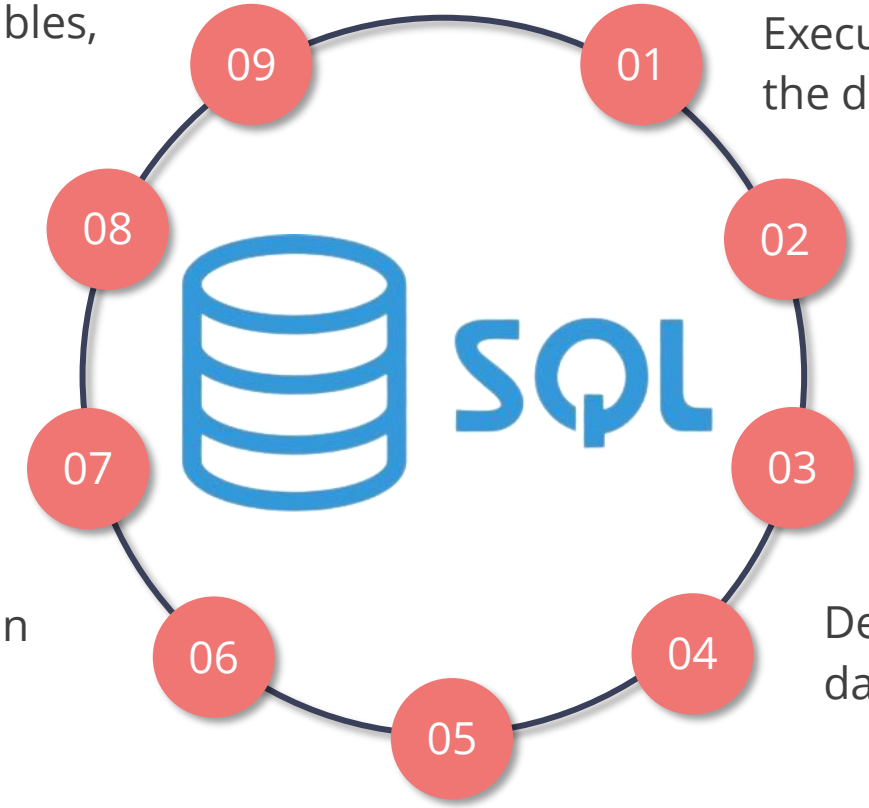
American National Standards Institute (ANSI) in 1986

International Organization for Standardization (ISO) in 1987

©Simplilearn. All rights reserved.

SQL

It is used to:



01 Execute queries against the database

02 Retrieve data from the database

03 Insert and update records into the database

04 Delete records in the database

05 Create new databases

06 Create a new table in the database

07 Create stored procedures in the database

08 Create views in the database

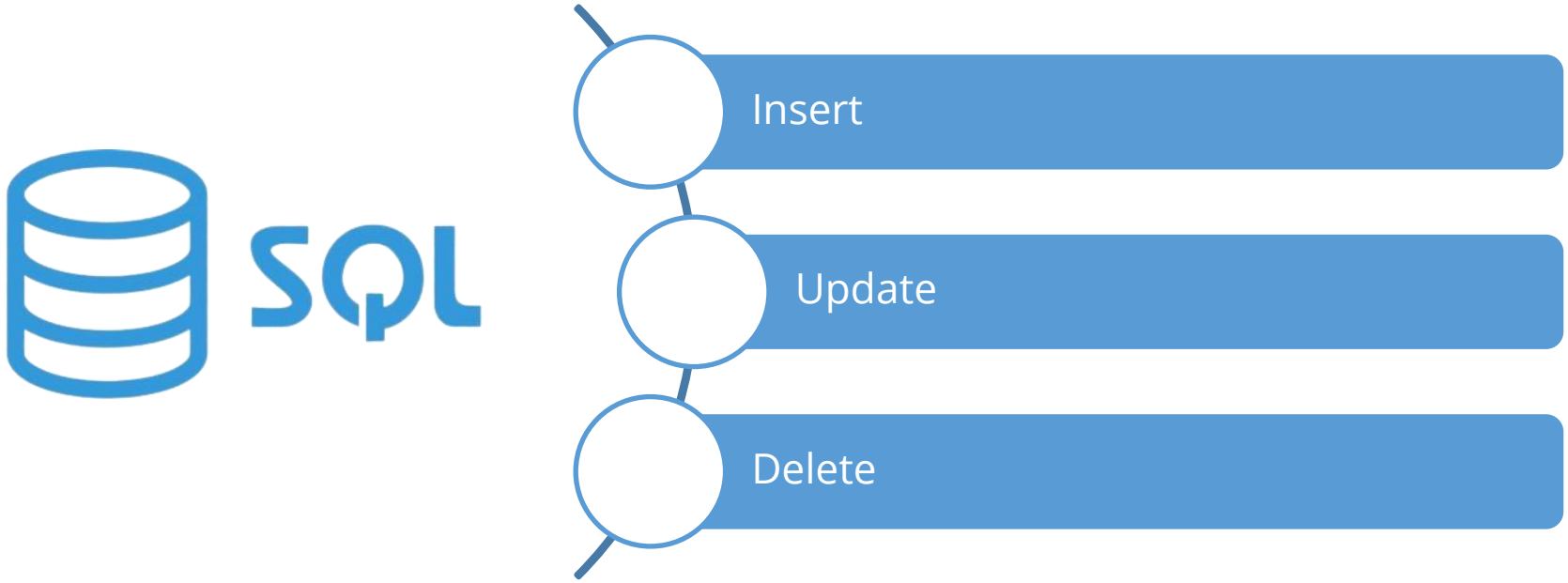
09 Set permissions on the tables, procedures, and views

©Simplilearn. All rights reserved.

SQL

SQL is based on ANSI or ISO standards with numerous SQL database versions.

SQL supports major commands, such as:



©Simplilearn. All rights reserved.

RDBMS

RDBMS stands for Relational Database Management System.

It is the foundation for SQL in all modern database systems, such as:

Microsoft SQL Server

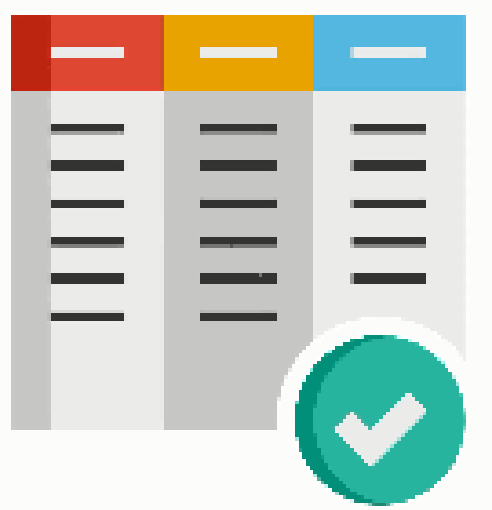
Microsoft Access

©Simplilearn. All rights reserved.

RDBMS

Data is stored in RDBMS tables which are database objects.

A table is a collection of data entries organized into columns and rows.



©Simplilearn. All rights reserved.

Creating JDBC Connections with MySQL

©Simplilearn. All rights reserved.

MySQL

It is a popular open-source relational database management system that uses SQL for accessing and manipulating databases.

©Simplilearn. All rights reserved.

MySQL: Key Components

Important MySQL database concepts include:

- Driver class
- Connection URL
- Username
- Password

©Simplilearn. All rights reserved.

MySQL: Key Components

Driver class

The driver class for the MySQL database is **com.mysql.jdbc.Driver.**

Connection URL

The connection URL for the MySQL database is **jdbc:mysql://hostname:port/databaseName.**

©Simplilearn. All rights reserved.

MySQL Parameters

Understanding MySQL Database Connection Parameters

jdbc:mysql://localhost:4400/techABC

jdbc	It is an application programming interface (API).
mysql	It is the database.
localhost	It is the server's name on which MySQL is running.
4400	It is the port number.
techABC	It is the database.

Note

Other database names can also be used. In such cases, replace the techABC with the required database name.

©Simplilearn. All rights reserved.

MySQL: Key Components

Username

The default username for the MySQL database is **root**.

Password

Password is given by the users when they install the MySQL database.

©Simplilearn. All rights reserved.

simplilearn

MySQL

Create a database using the following command:

```
create database techABC;
use techABC;
create table employee(id int(20), name varchar(40),
age int(50), email varchar(50), phone bigint(12));
```

Database Name

The root is both the username and the password.

©Simplilearn. All rights reserved.

MySQL: Syntax

Syntax to create a table:

```
import java.sql.*;
        public class Employee{
        public static void main(String args[]){
         try{
         Class.name("com.mysql.jdbc.Driver");
         Connection connection =
DriverManager.getConnection(
          "jdbc:mysql://localhost:4400/techABC",
<username>, <password>);
         Statement statement =
connection.createStatement();
```

©Simplilearn. All rights reserved.

MySQL: Example

```
ResultSet result = statement.executeQuery("select * from
employee");
         while(result.next())
         System.out.println(result.getInt(1)+"
"+result.getString(2)+" "+result.getString(3));
         connection.close();
        } catch(Exception error) {
System.out.println(error);
        }
      }
     }
```

This example will retrieve all records of the employee table.

©Simplilearn. All rights reserved.

MySQL Setup

To connect the Java application with the MySQL database, the mysqlconnector.jar file needs to be loaded.

mysqlconnector.jar file is loaded using the following two methods:

1

The first method is to paste the mysql-connector-java.Jar file in the jre/lib/ext folder.

2

The second is to set the classpath.

©Simplilearn. All rights reserved.

MySQL Setup

To paste the mysqlcon.jar file in the jre/lib/extt folder:

Step 01

Download the mysql-connector-java.jar file

Step 02

Go to the jre/lib/extt folder

Step 03

Paste the jar file there

©Simplilearn. All rights reserved.

MySQL Setup

There are two ways to configure the classpath:

Temporary

Permanent

©Simplilearn. All rights reserved.

MySQL Setup

Steps to configure temporary classpath:

Open the command prompt

↓

Execute this command

↓

C:>set classpath=<path-to-mysql-connector-java-7.0.8-bin.jar>;

©Simplilearn. All rights reserved.

MySQL Setup

Steps to configure permanent classpath:

Go to the environment variable

↓

Click on the new tab

↓

In the variable name, write the classpath

↓

In variable value, paste the path to the mysql-connector-java-7.0.8-bin.jar file, for example, C:\path\to\mysql-connector-java-7.0.8-bin.jar;

©Simplilearn. All rights reserved.

Configuring JDBC and Creating Connection to MySQL DB

Problem Statement:

You have been asked to configure JDBC and create a connection to a MySQL database by creating a Java project and executing the createConnection() method.

Outcome:

By configuring JDBC and creating a connection to a MySQL database in a Java project, you will learn to set up database connectivity and execute the createConnection() method. This knowledge is essential for building applications that interact with relational databases efficiently.

Note: Refer to the demo document for detailed steps: 01_Configuring_JDBC_and_Creating_Connection_to_MySQL_DB

ASSISTED PRACTICE

©Simplilearn. All rights reserved.

Assisted Practice: Guidelines

Steps to be followed are:

1. Create a Java project
2. Open the Google browser
3. Create a new project
4. Create an interface in com.example.cms
5. Create one more class and name it DB
6. Create a connection in the DAO.java file
7. Initialize the connection
8. Execute the createConnection method

©Simplilearn. All rights reserved.

Creating Table in DB and Implementing ORM with OOPS

Problem Statement:

You have been asked to create a customer model and a table in the database and implement ORM with OOPS.

Outcome:

By creating a customer model and a table in the database and implementing ORM with OOP principles, you will learn to map objects to database records. This enhances your ability to manage database operations through object-oriented code, promoting efficiency and maintainability.

Note: Refer to the demo document for detailed steps: 02_Creating_Table_in_DB_and_Implementing_ORM_with_OOPS

ASSISTED PRACTICE

©Simplilearn. All rights reserved.

Assisted Practice: Guidelines

Steps to be followed are:

1. Create a new class
2. Write a customer object in App.java
3. Open the terminal window
4. Declare a method for a customer in the DAO interface

©Simplilearn. All rights reserved.

JDBC Architecture

©Simplilearn. All rights reserved.

JDBC Architecture

It supports both two-tier and three-tier processing models for database access.

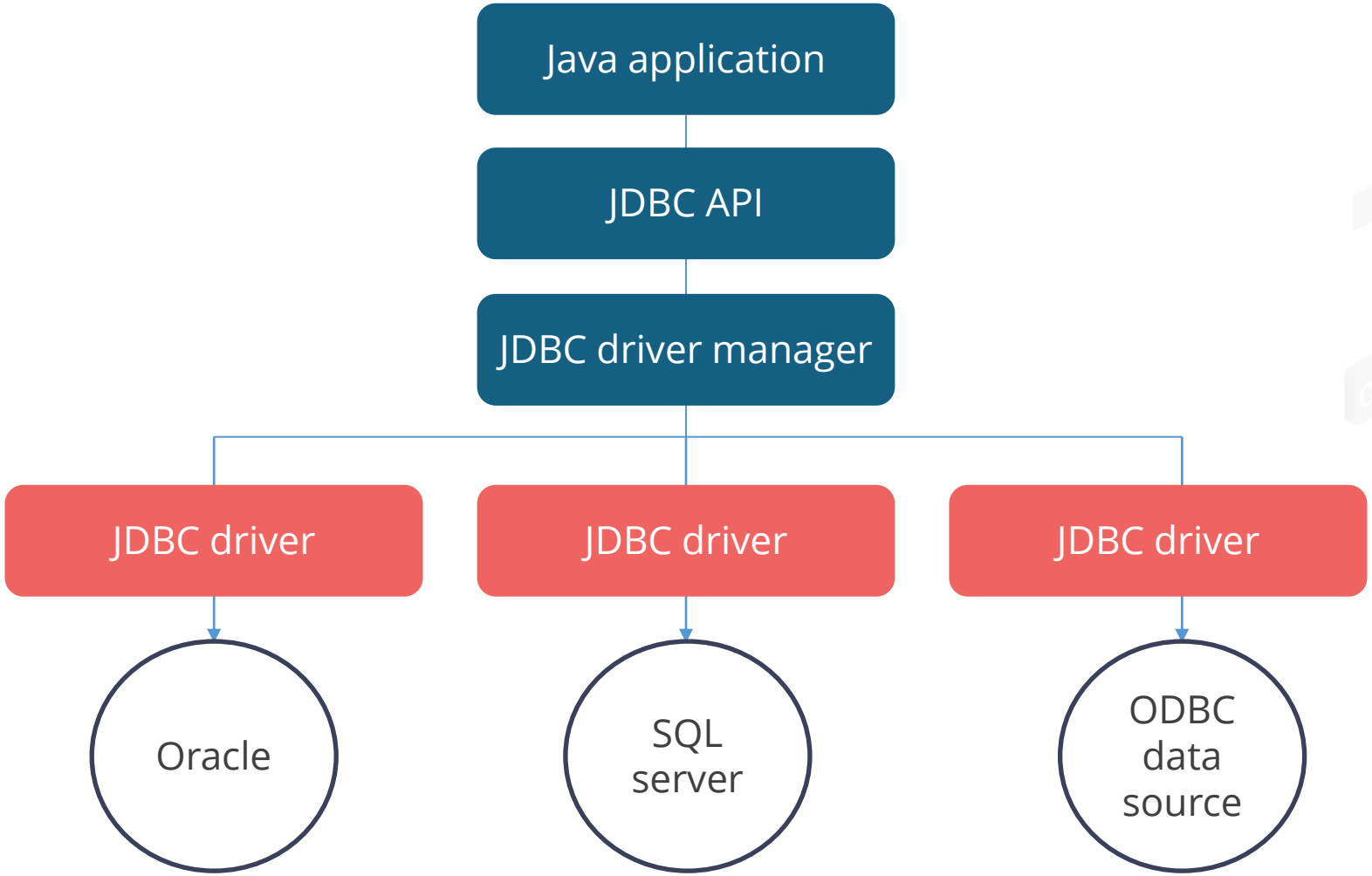
JDBC API → Provides the connection between the application and the JDBC manager

JDBC driver API → Supports the connection between the JDBC manager and the driver

©Simplilearn. All rights reserved.

JDBC Architecture

JDBC architectural diagram:

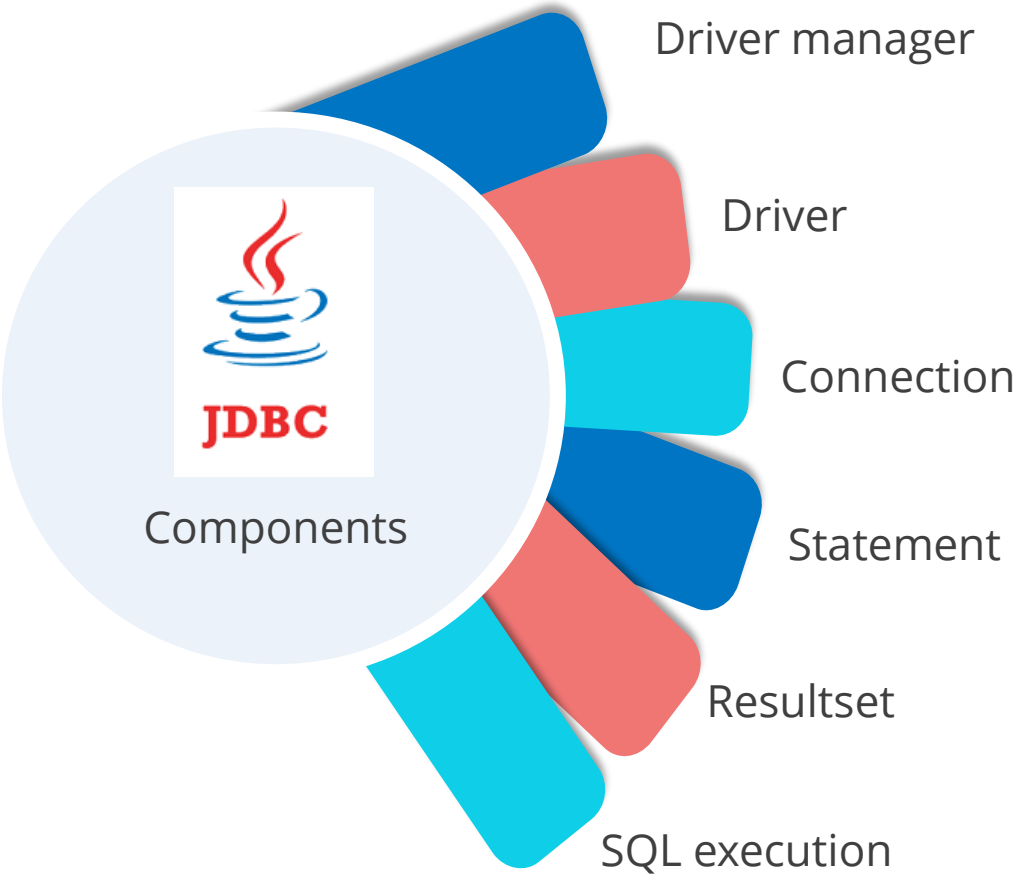


©Simplilearn. All rights reserved.

JDBC Components

©Simplilearn. All rights reserved.

JDBC Components

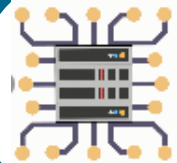


©Simplilearn. All rights reserved.

Driver Manager

It maintains a list of database drivers.

It uses the communication sub-protocol to match connection requests from Java applications with the appropriate database driver.



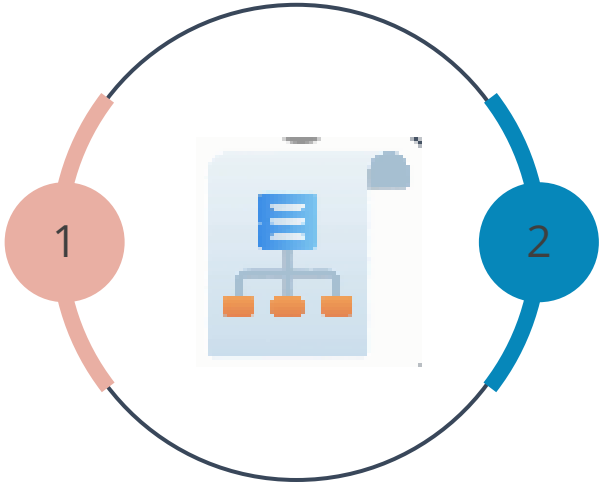
It recognizes a specific sub-protocol under the JDBC standard to connect to the database.

©Simplilearn. All rights reserved.

Driver

The functions of the Driver include:

1. Handles database server's communications via interface
2. Hides the details of internal mechanics of working with driver objects



Note

The user will use DriverManager objects to manage this Driver object.

©Simplilearn. All rights reserved.

Connection

This interface contains all the methods required to communicate with the database.

It represents communication context.

In other words, all communication with the database is through connection object only.

©Simplilearn. All rights reserved.

Statement

This interface is used to submit SQL statements to the database using the objects created.

It can also accept parameters and perform stored procedures.

©Simplilearn. All rights reserved.

ResultSet

ResultSet objects:

1

Retain the data returned from a database after using the statement objects to perform an SQL query

2

Operate as an iterator allowing one to loop through the data

©Simplilearn. All rights reserved.

SQLException

The SQLException class handles all errors that occur in a database application.

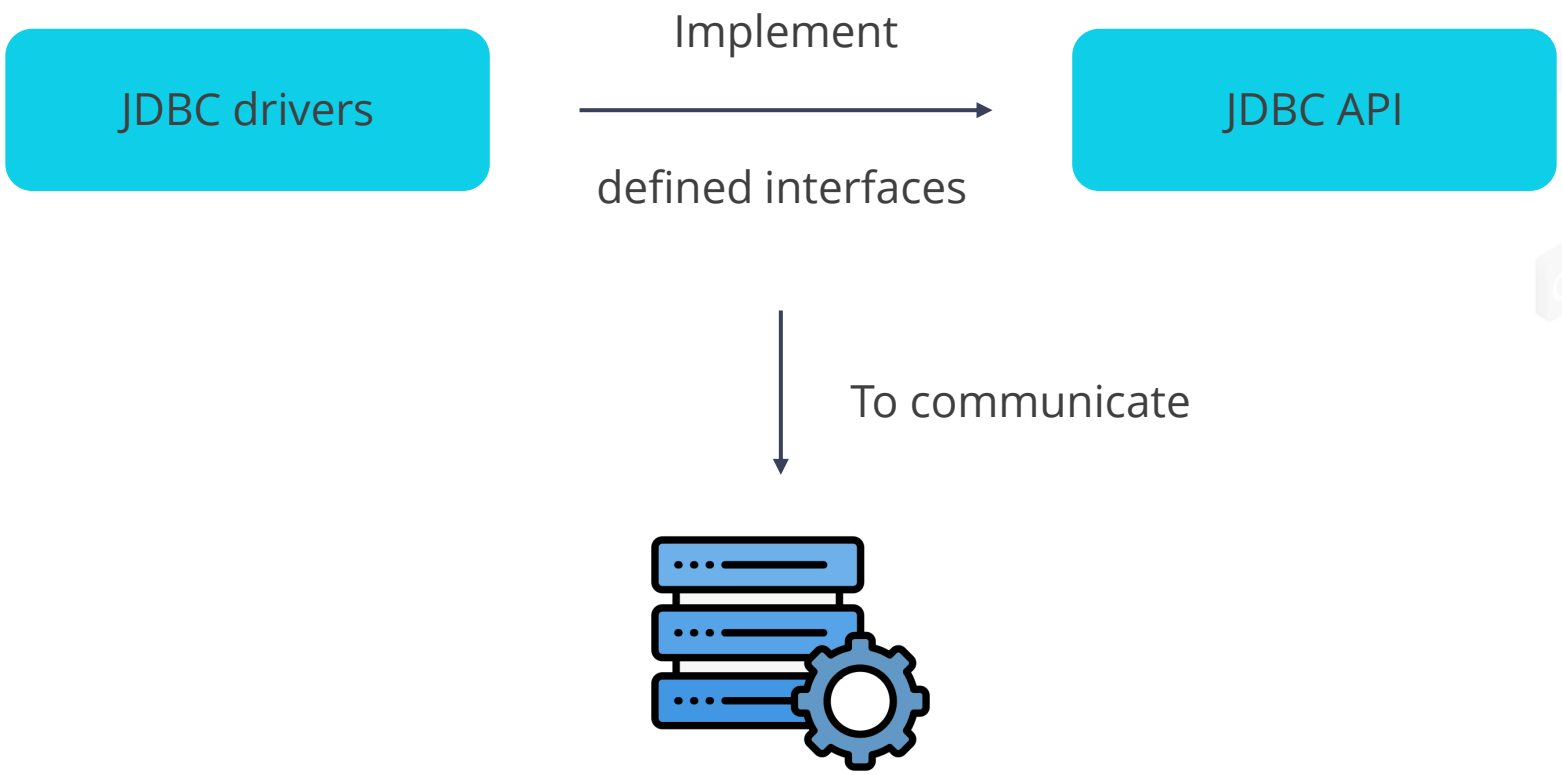
©Simplilearn. All rights reserved.

JDBC Drivers

©Simplilearn. All rights reserved.

JDBC Drivers

To communicate with a database, JDBC drivers implement defined interfaces in the JDBC API.

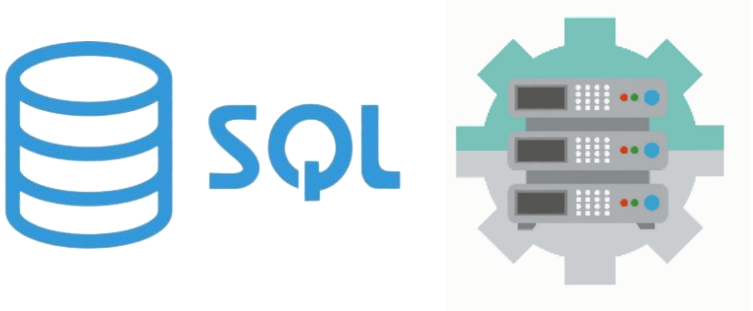


©Simplilearn. All rights reserved.

JDBC Drivers

Users can interact with the database using JDBC drivers.

Example:



Sending SQL or database instructions

Receiving →

Results in java

©Simplilearn. All rights reserved.

JDBC Drivers

The JDK's java.sql package contains several classes with defined behaviors.

The java.sql.Driver interface is implemented by third-party database drivers.

JDK's java.sql package

java.sql.Driver interface

©Simplilearn. All rights reserved.

JDBC Drivers

Java runs on a vast range of operating systems and hardware platforms.

JDBC driver implementation is divided into four categories:

Type 1 Driver

Type 2 Driver

JDBC

Type 4 Driver

Type 3 Driver

©Simplilearn. All rights reserved.

Type 1: JDBC-ODBC Bridge Driver

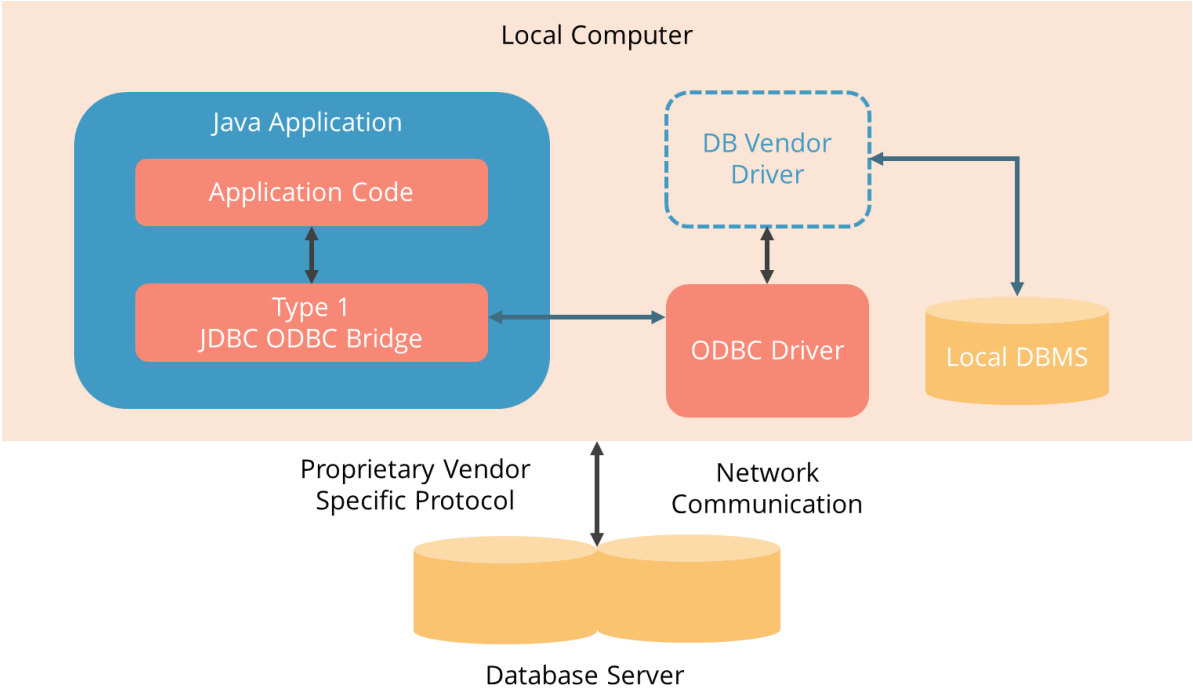
The JDBC bridge is used to connect to the ODBC drivers installed on each client machine.

To use a Type 1 driver, first set up Data Source Name, which represents the target database.

©Simplilearn. All rights reserved.

Type 1: JDBC-ODBC Bridge Driver

Type 1 driver was useful when Java first came out, as most databases only allowed ODBC access.

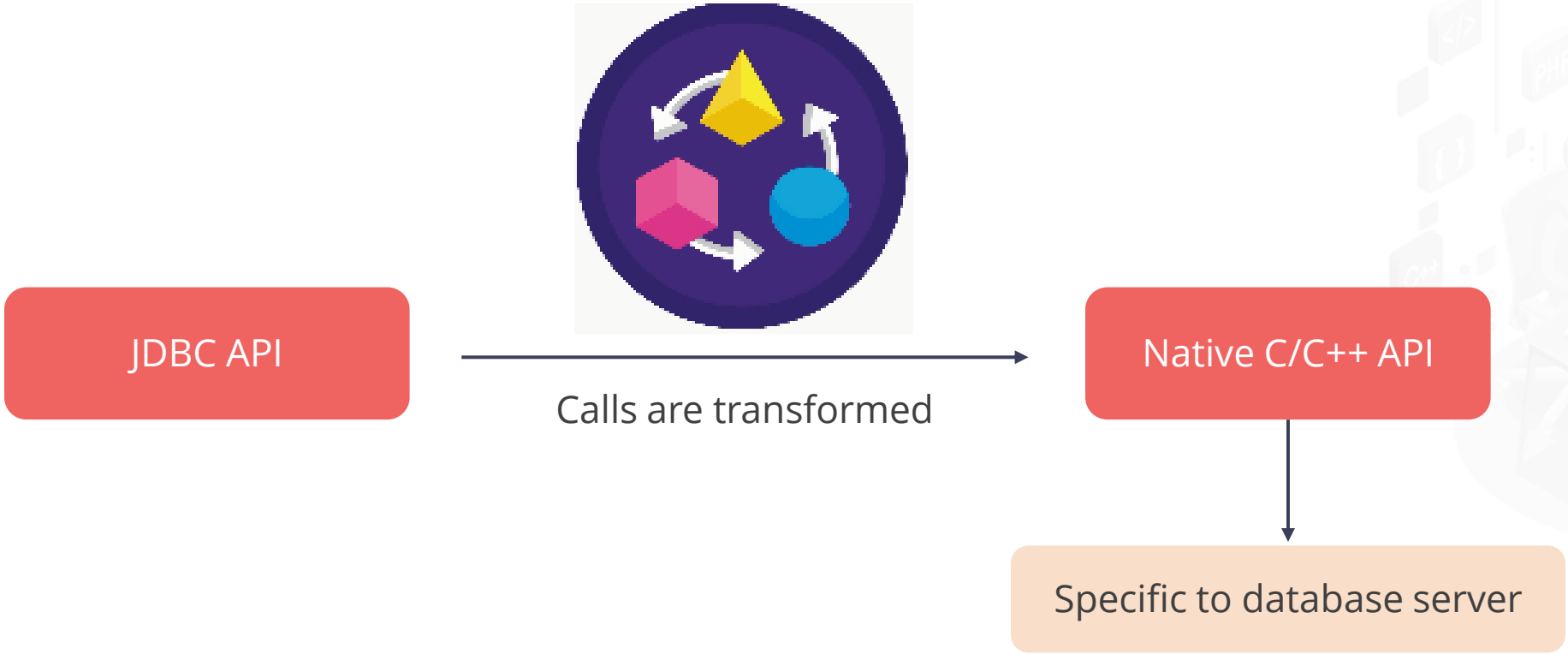


This type of driver is only suggested for experimental use or when no other option is available.

©Simplilearn. All rights reserved.

Type 2 : JDBC-Native API

In Type 2, the JDBC API calls are transformed into native C/C++ API calls that are specific to the database server.

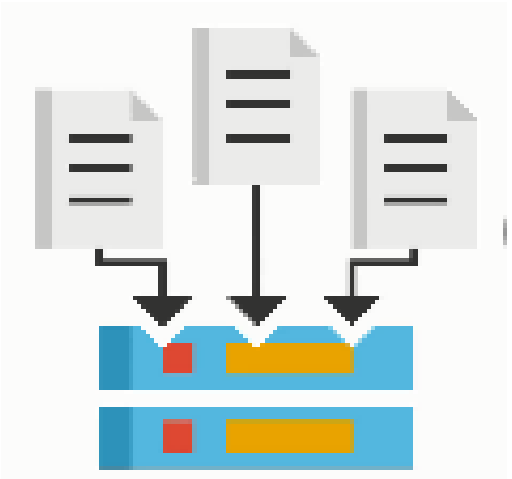


©Simplilearn. All rights reserved.

Type 2 : JDBC-Native API

Initially, database providers offer these drivers, which can be used in the same way as other JDBC-ODBC bridges.

The Type 2 driver is a vendor-specific driver that must be installed on all client computers.



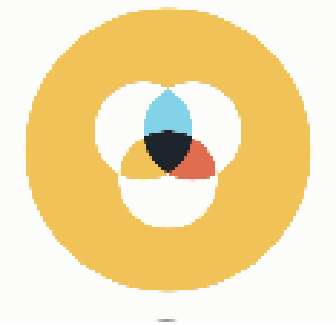
The native API should be modified if the database is changed.

©Simplilearn. All rights reserved.

Type 3: JDBC-Net Pure Java

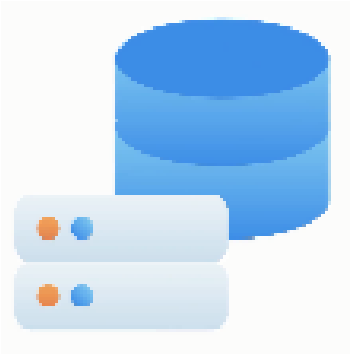
The Type 3 driver employs a three-tier strategy to access the database.

JDBC clients use conventional network sockets to connect with the middleware application server.



Three – tier strategy

Employed to →



Access Database

©Simplilearn. All rights reserved.

Type 3: JDBC-Net Pure Java

The web socket information is processed by the middleware application server.

This information is delivered to the database server in the request format required by the DBMS.

Web socket information

Processed by

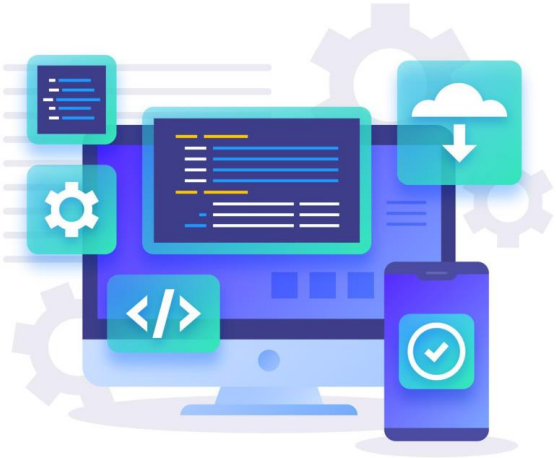
Middleware application server

©Simplilearn. All rights reserved.

Type 3: JDBC-Net Pure Java

Type 3 driver is adaptable and does not require the client to install any code.

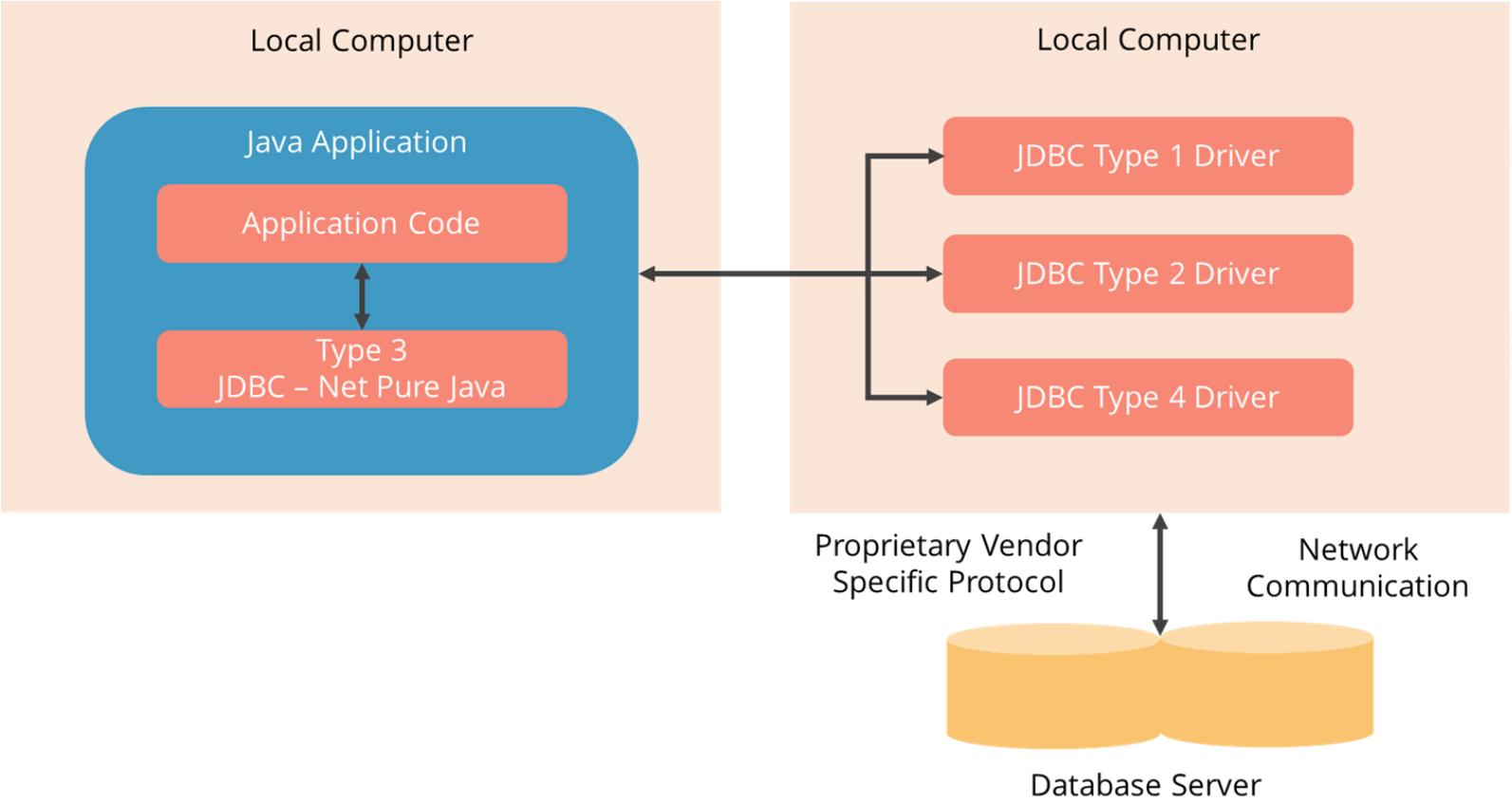
Typically, only a single driver can offer access to various databases.



©Simplilearn. All rights reserved.

Type 3: JDBC-Net Pure Java

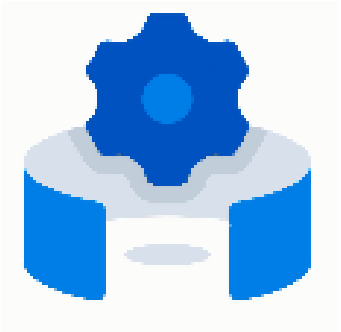
The application server acts as a JDBC proxy, allowing it to make calls on behalf of the client application.



©Simplilearn. All rights reserved.

Type 4: 100% Pure Java

The pure Java Type 4 driver interfaces directly with the vendor's database through a socket connection



Type 4 driver

Interfaces

Vendor's database

Directly through socket connection

©Simplilearn. All rights reserved.

Type 4: 100% Pure Java

It is the highest-performing database driver available.

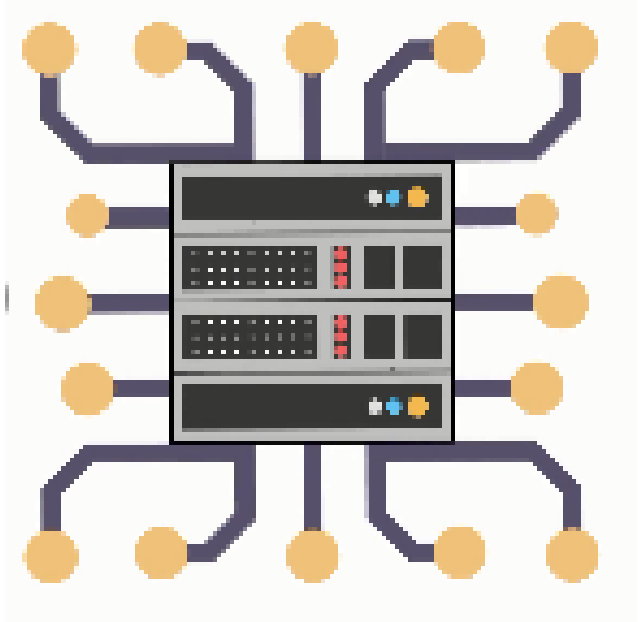
This driver is also quite adaptable and does not require any special software to be installed on client or server.

These drivers can also be dynamically downloaded.

©Simplilearn. All rights reserved.

Type 4: 100% Pure Java

The MySQL connector/j driver is a type 4 driver.



Database suppliers typically provide type 4 drivers due to the proprietary nature of their network protocols.

©Simplilearn. All rights reserved.

CRUD Operations Using JDBC

©Simplilearn. All rights reserved.

CRUD

CRUD is an acronym that originated from the world of computer programming languages. It stands for:

- Create
- Read
- Update
- Delete

©Simplilearn. All rights reserved.

Insert Statements Using Java JDBC

A program to insert statements using Java JDBC:

```
import java.sql.*;
 Class Student
{
public static void main(String[] args) throws ClassNotFoundException, SQLException
{
Class.name("oracle.jdbc.driver.OracleDriver");
Connection connection = DriverManager.getConnection("jdbc:oracle:thin:@localhost:4400:xe",
"System", "rose");
Statement statement = connection.createStatement();
int insertCount = statement.executeUpdate("insert into student  values (2004, 'jack',
4111)");
System.out.println(insertCount  + " stored successfully");
int insertCount = sts.executeUpdate("insert into student  values(2002, 'jack', 7011)");
System.out.println(insertCount + " more student data stored successfully to the MySQL
server ");
sts.close();
conn.close();
}}
```

©Simplilearn. All rights reserved.

Update Operations Using Java JDBC

The SQL statement for updating: Update set salary = salary + 7000.

```
Import java.sql.*;
 Class EmployeeSalary
{
public static void main (String args[]) throws ClassNotFoundException,
SQLException
{
Class.name ("oracle.jdbc.driver.OracleDriver");
Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:4400:techABC",
<username>, <password>);
Statement statement = connection.createStatement();
int updateCounts = statement.executeUpdate ("update employee  set salary = salary
+ 7000");
System.out.println(updateCounts + "rows are updated");
statement.close();
connection.close();
}
}
```

©Simplilearn. All rights reserved.

Delete Operations Using Java JDBC

The SQL statement is deleted from the account where empId = " + employee Id:

```
Import java.sql.*;
Import java.util.*;
Class Employee
 {public static void main (String args[]) throws ClassNotFoundException, SQLException
 {Scanner scanner = new Scanner (System.in);
System.out.println("Enter Employee id: ");
Int employeeId = scanner.nextInt();
Class.name("oracle.jdbc.driver.OracleDriver");
Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:4400:techABC", <username>,
<password>);
Statement statement = connection.createStatement();
Int c = statement.executeUpdate("delete from employee where empId =" + employeeId);
If ( c==0 )
System.out.println( "employee records. doesnot exist corresponds to employee id");
else
System.out.println( "employee deleted successfully from the MySQL Server" );
statement.close();
connection.close();
} }
```

©Simplilearn. All rights reserved.

Retrieve Operations Using Java JDBC

If the employee id exists, it should display the same.

```
Import java.sql.*;
Import java.util.*;
Class exmp
{
public static void main (String[] args) throws ClassFoundException, SQLException
{
Scanner scanner = new Scanner (System.in);
System.out.println("Enter Employee Id ");
Int employeeId = scanner.nextInt();
Class.name ("oracle.jdbc.driver.OracleDriver");
Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:4400:em", <username> ,
<password>);
Statement statement= = conn.createStatement();
ResultSet  result = statement.executeQuery("select * from employee where
employeeId=" + employeeId);
```

©Simplilearn. All rights reserved.

Retrieve Operations Using Java JDBC

If the employee id exists, it should display the same.

```
If (result.next ())
{
System.out.println("Emp. Id:  " + rss.getInt(1));
System.out.println("Emp. Name: " + rss.getString(2));
System.out.println("Emp. Salary : " + rss.getFloat(3));
System.out.println("Emp Email : " + rss.getString(4));
}
else
System.out.println("employee does not exist corresponds to entered
id");
result.close();
statement.close();
connection.close();
}
}
```

©Simplilearn. All rights reserved.

Performing CRUD Operations

Problem Statement:

You have been asked to work on create, read, update, and delete operations on the created database.

Outcome:

By performing create, read, update, and delete (CRUD) operations on the created database, you will learn to interact with and manage database records effectively. This is essential for developing robust applications that require persistent data storage and manipulation.

Note: Refer to the demo document for detailed steps: 03_Performing_CRUD_Operations

ASSISTED PRACTICE

©Simplilearn. All rights reserved.

Assisted Practice: Guidelines

Steps to be followed are:

1. Open the Terminal Emulator
2. Write an insert operation in Eclipse IDE
3. Create a SQL query for the insert operation
4. Write an update operation
5. Perform the get all customer operations
6. Perform the delete operation

©Simplilearn. All rights reserved.

Writing CRUD Operations with PreparedStatement API

Problem Statement:

You have been asked to implement the PreparedStatment API with JDBC to write CRUD operations.

Outcome:

By implementing the PreparedStatement API with JDBC to write CRUD operations, you will learn to perform database interactions securely and efficiently. This approach helps prevent SQL injection attacks and improves the performance of your database operations.

Note: Refer to the demo document for detailed steps: 04_Writing_CRUD_Operations_with_PreparedStatement_API

ASSISTED PRACTICE

©Simplilearn. All rights reserved.

Assisted Practice: Guidelines

Steps to be followed are:

1. Perform the Create operation
2. Perform the Update operation
3. Perform the Delete operation
4. Perform the getAllCustomer operation

©Simplilearn. All rights reserved.

Batch Processing Transaction

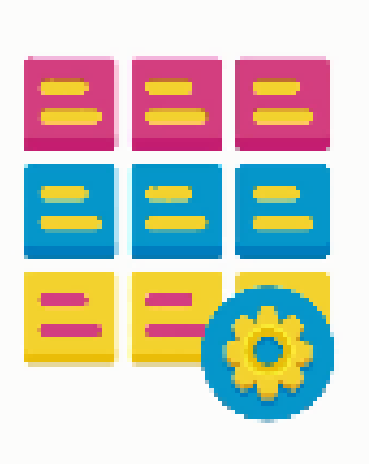
©Simplilearn. All rights reserved.

Batch Process

The batch process allows users to collect similar SQL statements into a batch.

A batch is submitted to the database with a single call.

SQL statements

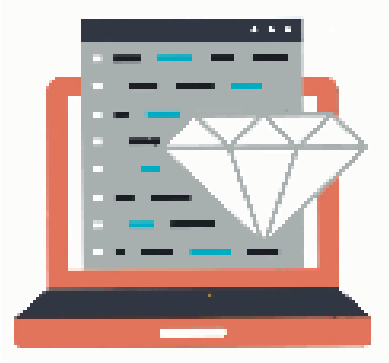


Batch

©Simplilearn. All rights reserved.

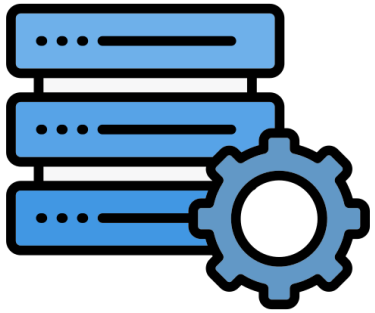
Batch Process

Submitting multiple SQL statements simultaneously to the database can minimize communication overhead and enhance performance.



SQL statements

Submitting to



Database

Reduces communication overhead and improves speed

©Simplilearn. All rights reserved.

simplilearn

Batch Process

To determine whether the target database supports batch update processing, use the **DatabaseMetaData.supportsBatchUpdates()** method.

If the JDBC driver supports this feature, this method returns true.

©Simplilearn. All rights reserved.

Batch Process

Single query strings are added to the Batch pool using the **addBatch()** method of the statement, Prepared statement, and callable statement.

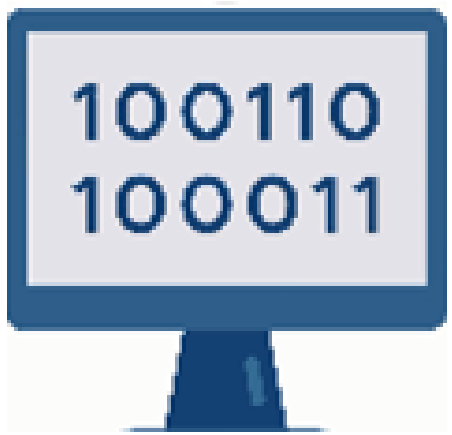
The executeBatch() method starts the execution of all the statements in a batch.

©Simplilearn. All rights reserved.

Batch Process

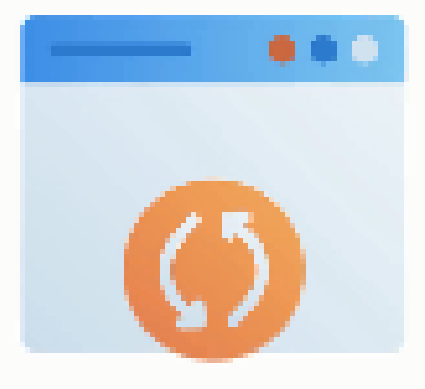
Every element of the Array represents the update count for the respective Update statement.

The executeBatch() returns an array of numbers.



Element of array

Represents →



Update statement

©Simplilearn. All rights reserved.

Batch Process

The clearBatch() method can be used to add and delete statements from the batch for processing.

All the statements inserted with the addBatch() method are removed using this method.

clearBatch() → Uses

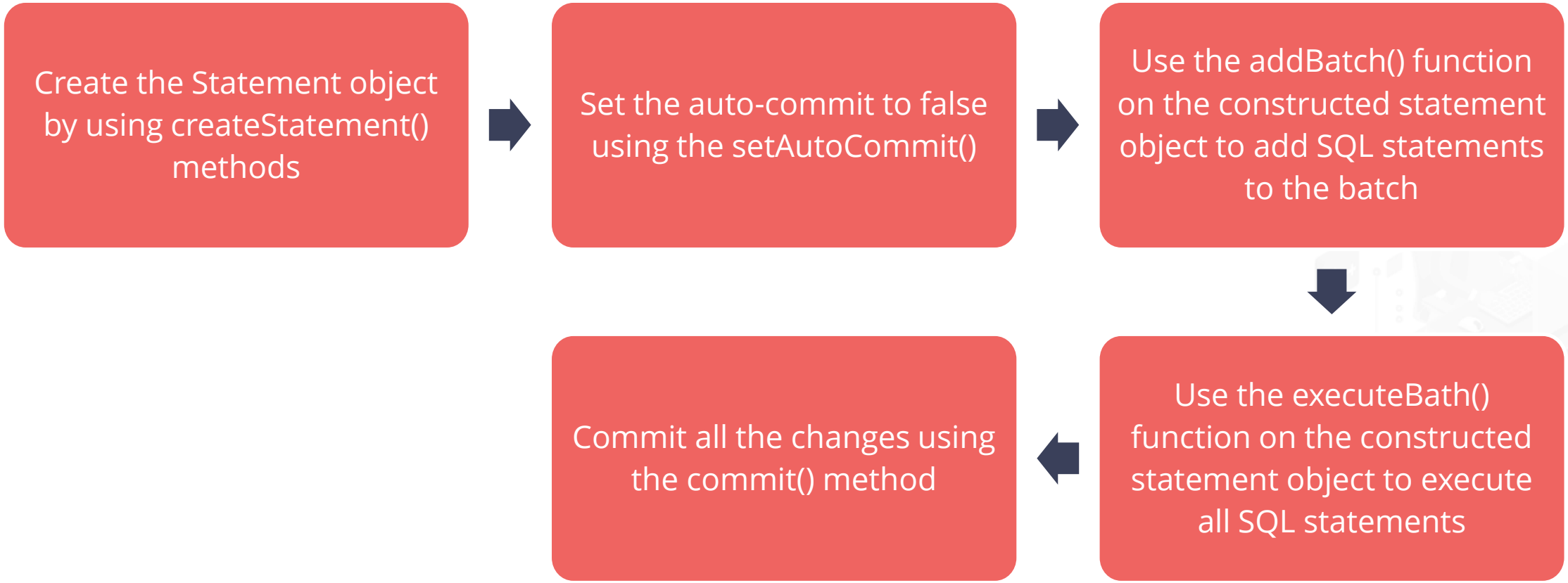
Note

Users cannot pick and choose statements to eliminate.

©Simplilearn. All rights reserved.

Batch Process: Steps

The steps used for Batch Processing with the Statement Object are as follows:



©Simplilearn. All rights reserved.

Batch Process: Example

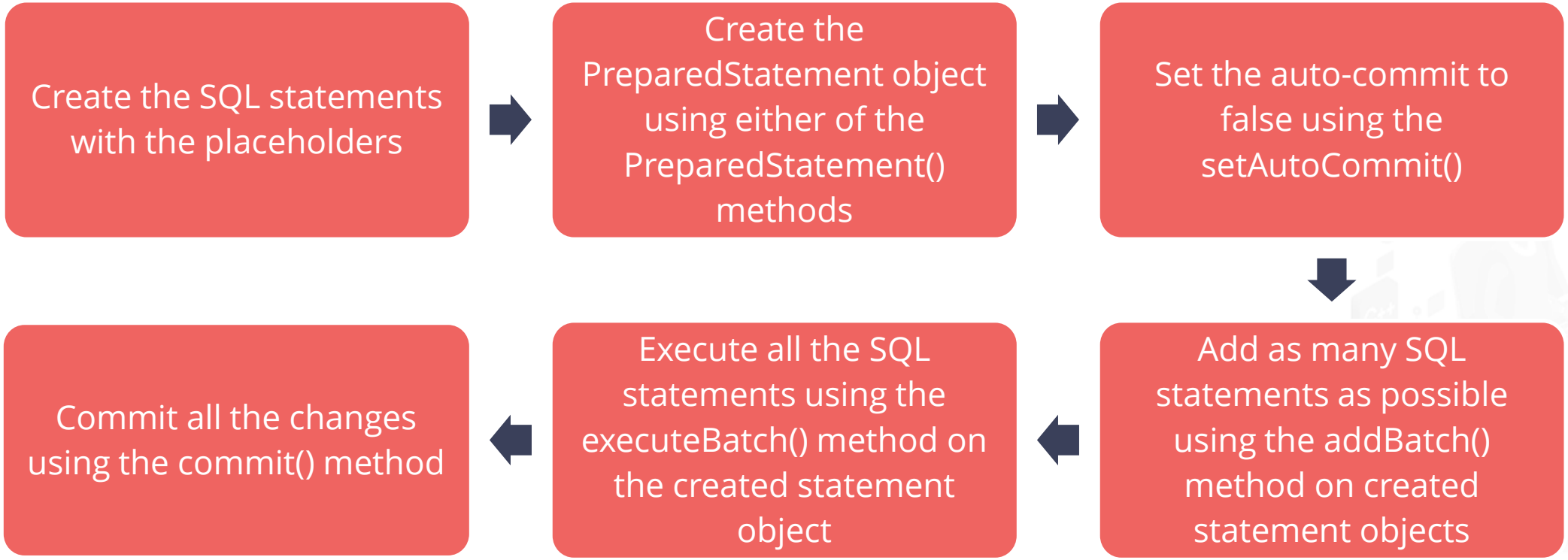
Example of a Batch Update using the Statement Object:

```
Statement statement = connection.createStatement();
connection.setAutoCommit(false);
String sqlQuery = " INSERT INTO Employee ( id, first, last, age,
email ) " +  " VALUES ( 12, 'John' , 'Peterson',  21,
'john@example.com' ) ";
statement.addBatch(sqlQuery);
String sqlQuery= " INSERT INTO Employee ( id, first, last, age,
email ) " + " VALUES ( 25, 'Robin' , '', 45, 'robin@example.com'
)";
statement.addBatch(sqlQuery);
String sqlQuery = " UPDATE Employee SET
email='robin.201@example.com' " + " WHERE id = 25 " ;
statement.addBatch(sqlQuery);
Int[] queryCount = statement.executeBatch();
connection.commit();
```

©Simplilearn. All rights reserved.

Batch Process: Steps

The steps to be used for Batch Processing with the PreparedStatement Object:



©Simplilearn. All rights reserved.

Batch Process: Example

```
String queryString = " INSERT INTO Employee ( id, first, last, age, email ) " +
" VALUES ( ?, ?, ?, ?, ? ) ";
PreparedStatement  pStatement  = connection.PreparedStatement(SQL);
connection.setAutoCommit(false);
pStatement.setInt ( 1, 100);
pStatement.setString( 2, "Fionna" );
pStatement.setString( 3, "Jackson" );
pStatement.setInt( 4, 40 );
pStatement.setString( 5, "fionna@example.com" );
pStatement.addBatch();
pStatement.setInt( 1, 101 );
pStatement.setString( 2, "Michael" );
pStatement.setString( 3, "Jackson" );
pStatement.setInt( 4, 41 );
pStatement.setString( 5, "jackson@example.com" );
pStatement.addBatch();
...
Int[] rowsCount= pStatement.executeBatch();
connection.commit();
```

©Simplilearn. All rights reserved.

Executing Batch Processing

Problem Statement:

You have been asked to explore batch processing and perform multiple SQL statements in a batch**.**

Outcome:

By exploring batch processing and performing multiple SQL statements in a batch, you will learn to execute several database operations in a single batch efficiently. This technique improves performance and reduces the overhead associated with individual database transactions.

Note: Refer to the demo document for detailed steps: 05_Executing Batch Processing

ASSISTED PRACTICE

©Simplilearn. All rights reserved.

Assisted Practice: Guidelines

Steps to be followed are:

1. Create a table for employee details
2. Create a method for batch execution
3. Use a Prepared statement
4. Use mixed SQL statements in a batch

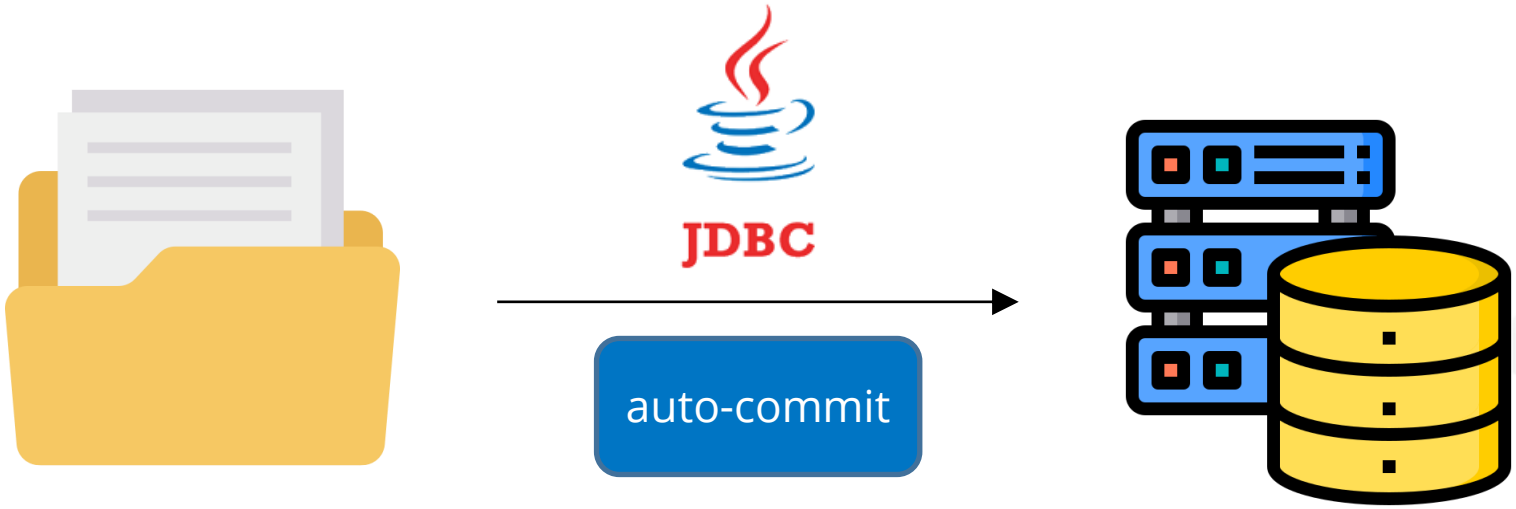
©Simplilearn. All rights reserved.

Commit and Rollback

©Simplilearn. All rights reserved.

Commit

It is used to save all changes made during the current transaction to the database, making them permanent.



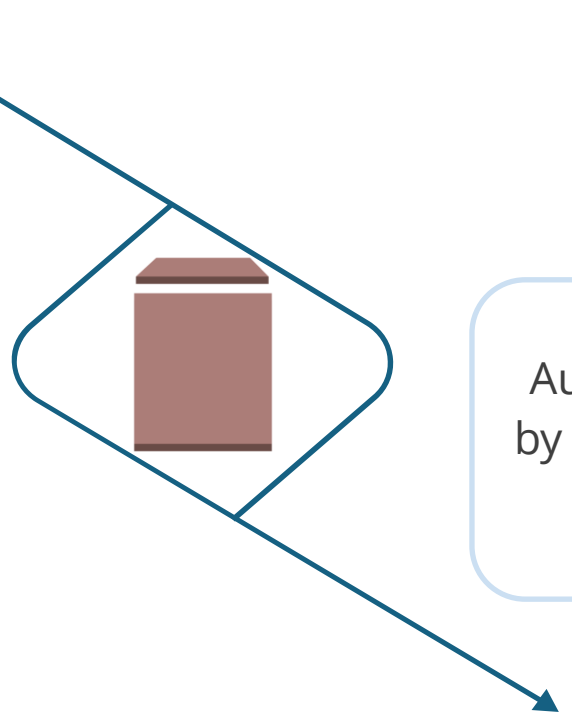
If the JDBC Connection is set to auto-commit, every Query Statement is committed to the database upon completion.

©Simplilearn. All rights reserved.

Commit

setAutoCommit() function of the Connection object is used to enable the manual transaction.

Auto-commit can be disabled by passing the boolean false to setAutoCommit().



Auto-commit can be enabled by passing the boolean true to setAutoCommit().

©Simplilearn. All rights reserved.

Commit

Key reasons to disable auto-commit are:

To improve performance and maintain the integrity of corporate processes

To be able to make use of distributed transactions

©Simplilearn. All rights reserved.

Commit

Code to disable auto-commit:

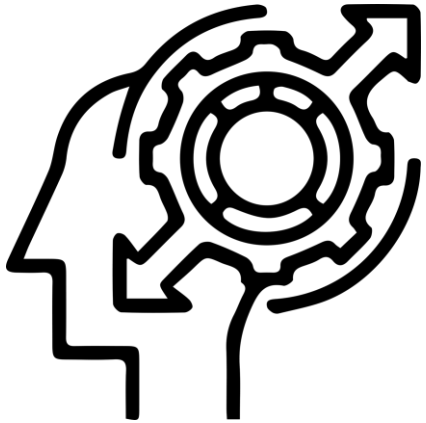
```
connection.setAutoCommit( false );
```

©Simplilearn. All rights reserved.

Transactions

It is a sequence of one or more SQL operations executed as a single unit of work.

If all operations succeed, the transaction is committed using commit.



If any operation fails, the transaction can be reverted using rollback, ensuring database integrity.

©Simplilearn. All rights reserved.

Commit

Once the changes are done, call the commit() method on the connection object to perform a commit.

```
connection.commit();
```

To rollback updates to the database made using the Connection named connection, use this code:

```
connection.rollback();
```

©Simplilearn. All rights reserved.

simplilearn

Commit: Syntax

The syntax for using commit and rollback:

```
try {
// assume a valid connection to the  object conn
connection.setAutoCommit(false);
Statement statement = connection.createStatement();
String queryString= " INSERT INTO Employee " + " VALUES (
102, 32, 'David', 'Jacklin' )";
stmt.executeUpdate(queryString);
// submit a malformed SQL statement that breaks
String queryString = " INSERT INTO Emp " + " VALUES (
270, 35, 'Jack', 'Singh', 'jack@example.com')";
statment.executeUpdate(queryString);
// if there is no error
connection.commit();
}
catch( SQLException  sqlError ) {
// if there is any error
connection.rollback();
}
```

©Simplilearn. All rights reserved.

Using Savepoints

Additional transactional control is available with the new and updated JDBC 3.0 Savepoint interface.

JDBC

Version 3.0

ORACLE PL/SQL

©Simplilearn. All rights reserved.

Using Savepoints

Establishing the savepoint means defining the transaction's logical rollback point.

If an error occurs after the savepoint → Use the rollback method to restore the database to its previous state

©Simplilearn. All rights reserved.

Using Savepoints

Functions helping in the management of Savepoints:

setSavepoint

This method creates a new Savepoint and returns a Savepoint object.

releaseSavepoint

In the scoring step, the scheduler ranks the remaining nodes to choose the most suitable pod placement.

The Savepoint generated by the setSavepoint() function is usually an object.

©Simplilearn. All rights reserved.

Using Savepoints

Example:

```
try {
// assume the valid connection object conn
conn.setAutoCommit(false);
Statement statement = connection.createStatement();
// set the Savepoint
Savepoint savePoint1 = conn.setSavepoint(“ Savepoint1” );
String queryString = " INSERT INTO Employee " + " VALUES
( 107, 34, 'Rita', ‘Sam’, null)";
statement.excuteUpdate(queryString);
// submit a malformed SQL statement that breaks
String queryString = " INSERT INTO Emp " + " VALUES (
200, 45, 'Harry', 'Shino', 'harry@example.com')";
statement.excuteUpdate(queryString);
// if no errors occur, then commit the changes
connection.commit();
}
```

©Simplilearn. All rights reserved.

Using Savepoints

Example:

```
catch( SQL Exception sqlError )
{
// if there is any error
connection.rollback( savePoint1 );
}
```

In this example, none of the INSERT statements succeeds, and everything will be rolled back.

©Simplilearn. All rights reserved.

simplilearn

Implementing Transaction Management

Problem Statement:

You have been asked to work on transaction management to execute all the SQL statements together.

Outcome:

By working on transaction management to execute all SQL statements together, you will learn to ensure data integrity and consistency. This skill is crucial for handling multiple operations as a single unit, allowing you to commit or roll back changes based on the success of the entire transaction.

Note: Refer to the demo document for detailed steps: 06_Implementing_Transaction_Management

ASSISTED PRACTICE

©Simplilearn. All rights reserved.

Assisted Practice: Guidelines

Steps to be followed are:

1. Create new table name orders
2. Create an execution Transaction method
3. Call the created method and check the output
4. Use the auto-commit feature
5. Use insert and update operations
6. Use the delete operation

©Simplilearn. All rights reserved.

Handling SQLException

©Simplilearn. All rights reserved.

SQLException

If JDBC encounters an issue while interacting with the data source, it will raise an SQLException.

The SQLException object contains information that can help determine the cause of the error.

©Simplilearn. All rights reserved.

SQLException

It retrieves the String object by calling the method that contains the description.

A description of the error

```
SQLException.getMessage()
```

©Simplilearn. All rights reserved.

SQLException

An SQL state code

The ISO, ANSI, and the Open Group (X/Open) have standardized the code and its meanings.

The five alphanumeric characters make up the String object.

```
SQLException.getSQLState
function
```

©Simplilearn. All rights reserved.

SQLException

Error code:

It is an integer value that identifies the SQLException instance that was thrown because of the problem.

Its value and meaning are implementation-dependent and could reflect the underlying data source's error code.

```
SQLException.getNextException
```

©Simplilearn. All rights reserved.

SQLException

A cause

An SQLException instance can be thrown if a causal relationship exists between it and one or more throwable objects.

Recursively invoke the SQLException method until a null value is returned

©Simplilearn. All rights reserved.

SQLException

A reference to any chained exceptions

The exceptions are referenced through this chain if there are several errors.

```
SQLException.getNextException
```

©Simplilearn. All rights reserved.

Retrieving Exceptions

The syntax to retrieve exceptions:

```
public static void printSQLException(SQLException
exception) {
   for (Throwable ee : exception) {
        if (ee instanceof SQLException) {
            if (ignoreSQLException(
                ((SQLException)ee).
                getSQLState()) == false) {
                ee.printStackTrace(System.err);
                System.err.println("SQLState: " +
                    ((SQLException)e).getSQLState());
                System.err.println("Error Code: " +
                    ((SQLException)e).getErrorCode());
                System.err.println("Message: " +
e.getMessage());
                Throwable t = exception.getCause();
                while(t != null) {
                    System.out.println("Cause: " + t);
                    t = t.getCause();
                }
            }           }       }       }
```

©Simplilearn. All rights reserved.

simplilearn

Warnings

Warnings do not stop the program's execution; instead, they notify the user that something did not go as expected.

SQLWarning objects are subclasses of SQLException that handle database access warnings.

©Simplilearn. All rights reserved.

Warnings

A warning can also be reported on the:

Connection object

Statement object

Resultset object

Each of these classes has a getWarnings method.

©Simplilearn. All rights reserved.

Warnings

Any warnings generated by a preceding statement are cleared when the statement is executed automatically, preventing accumulation.

Demonstrate how to obtain comprehensive information on any warnings reported on Statement or ResultSet objects using the provided method

©Simplilearn. All rights reserved.

Warnings

The syntax is as shown below:

```
Public static void getWarningsFromResultSet(ResultSet rs)
Throws SQLException {
JDBCUtilities.printWarningsFromStatement(rs.getWarnings());
}
Public static void getWarningsFromStatement(Statement stmt)
Throws SQLException{
JDBCUtilities.printWarnings(stmt.getWarnings());
}
Public static void printWarnings(SQL Warning warning)
Throws SQLException {
If (warning != null){
System.out.println('\n--------Warning-------\n");
While (warning != null){
System.out.println("Message:" +warning.getMessage());
System.out.println("SQLSate:" + warning.getSQLSate());
System.out.println(" error code");
System.out.println(warning.getErrorCode());
System.out.println(" ");
Warning = warning.getNextWarning();
}        }
```

©Simplilearn. All rights reserved.

simplilearn

Warnings

The most common warning is the DataTruncation warning, which is a subclass of SQLWarning.

01004

©Simplilearn. All rights reserved.

DataTruncation

It is a method used to determine whether column or parameter data was truncated.

It also determines if the truncation occurred during a read or write operation and how many bytes were actually sent.

©Simplilearn. All rights reserved.

Creating Stored Procedures

Problem Statement:

You have been asked to explore stored procedures and execute them using the JDBC API called callable statements.

Outcome:

By exploring stored procedures and executing them using the JDBC API called CallableStatement, you will learn to encapsulate complex database operations for reuse and efficiency. This approach enhances modularity and security in your database interactions, improving overall application performance and maintainability.

Note: Refer to the demo document for detailed steps: 07_Creating_Stored_Procedures

ASSISTED PRACTICE

©Simplilearn. All rights reserved.

Assisted Practice: Guidelines

Steps to be followed are:

1. Create a new table
2. Create a stored procedure
3. Create a method to execute a stored procedure
4. Create a stored procedure to read data
5. Create a method to call a stored procedure to read data

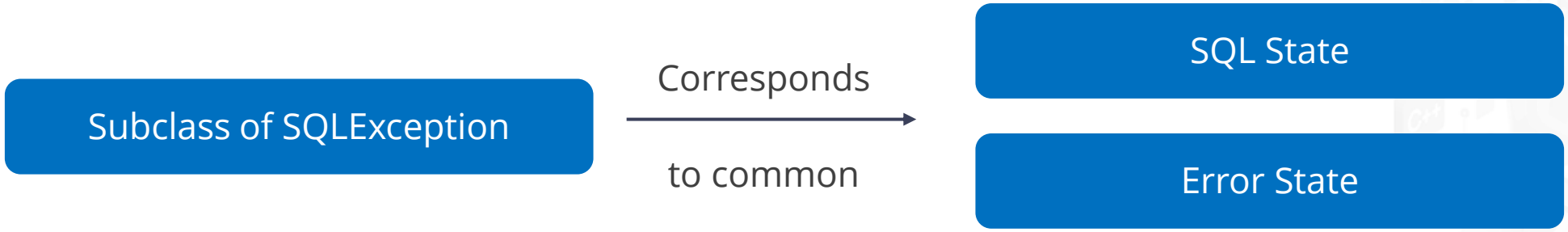
©Simplilearn. All rights reserved.

Categorizing SQLException

©Simplilearn. All rights reserved.

Categorizing SQLException

A subclass of the SQLException that corresponds to a common SQL state or a common error state is a specific SQL state class value not associated with a JDBC driver.



It allows for writing more portable error-handling code.

©Simplilearn. All rights reserved.

Categorizing SQLException

Subclasses of the supplied classes contain three exceptions:

SQLNonTransient Exception

SQLTransient Exception

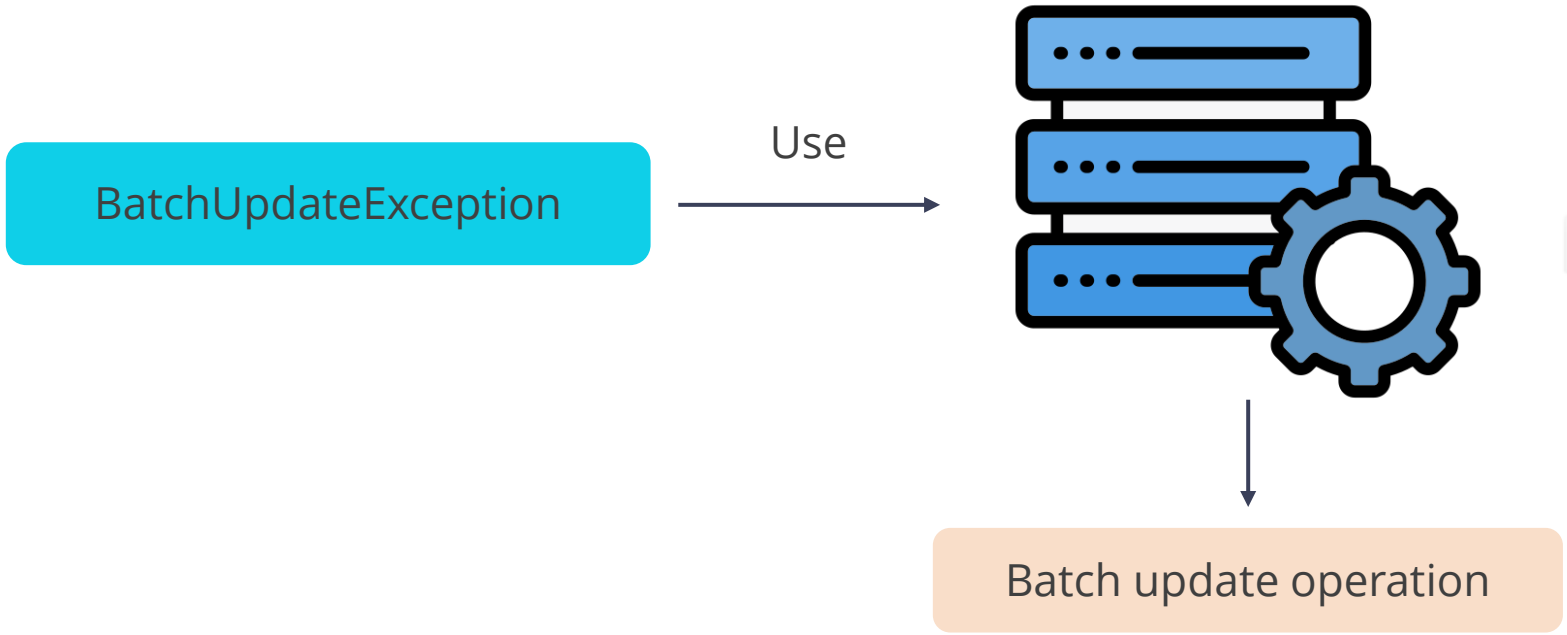
SQLRecoverable Exception

Subclasses of the SQLException can also be written in several ways.

©Simplilearn. All rights reserved.

BatchUpdateException

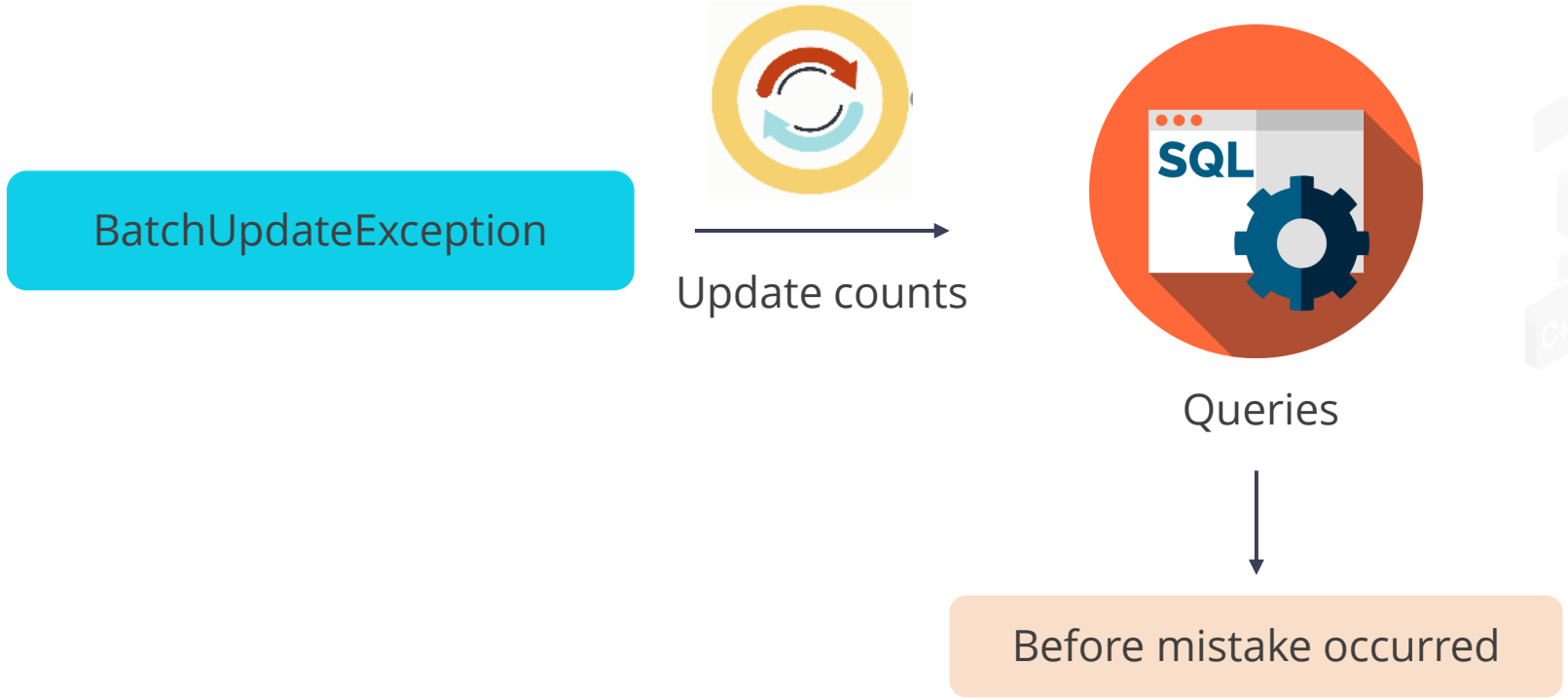
It is used when an error occurs during a batch update operation.



©Simplilearn. All rights reserved.

BatchUpdateException

It also provides the update counts for all the queries that were performed before the mistake occurred.



©Simplilearn. All rights reserved.

SQLClientInfoException

It is used when one or more client information properties on the connection are not set.

SQLClientInfoException

Used

Clients

Information properties on the connection are not set.

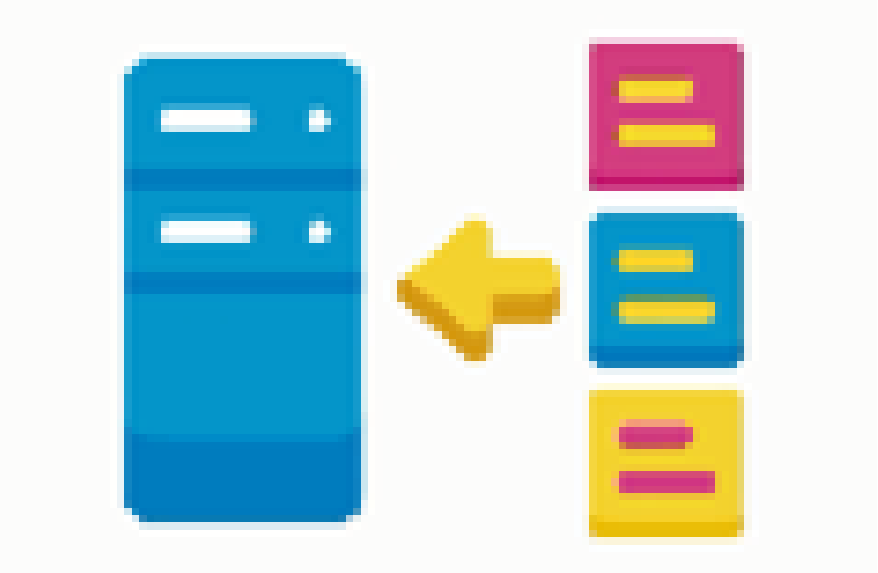
©Simplilearn. All rights reserved.

RowSet Objects

©Simplilearn. All rights reserved.

RowSet Objects

It is an object that encapsulates a set of rows from JDBC or other data sources.



RowSet objects are derived from the ResultSet interface.

©Simplilearn. All rights reserved.

RowSet Objects

They are unique because they bring additional features such as:

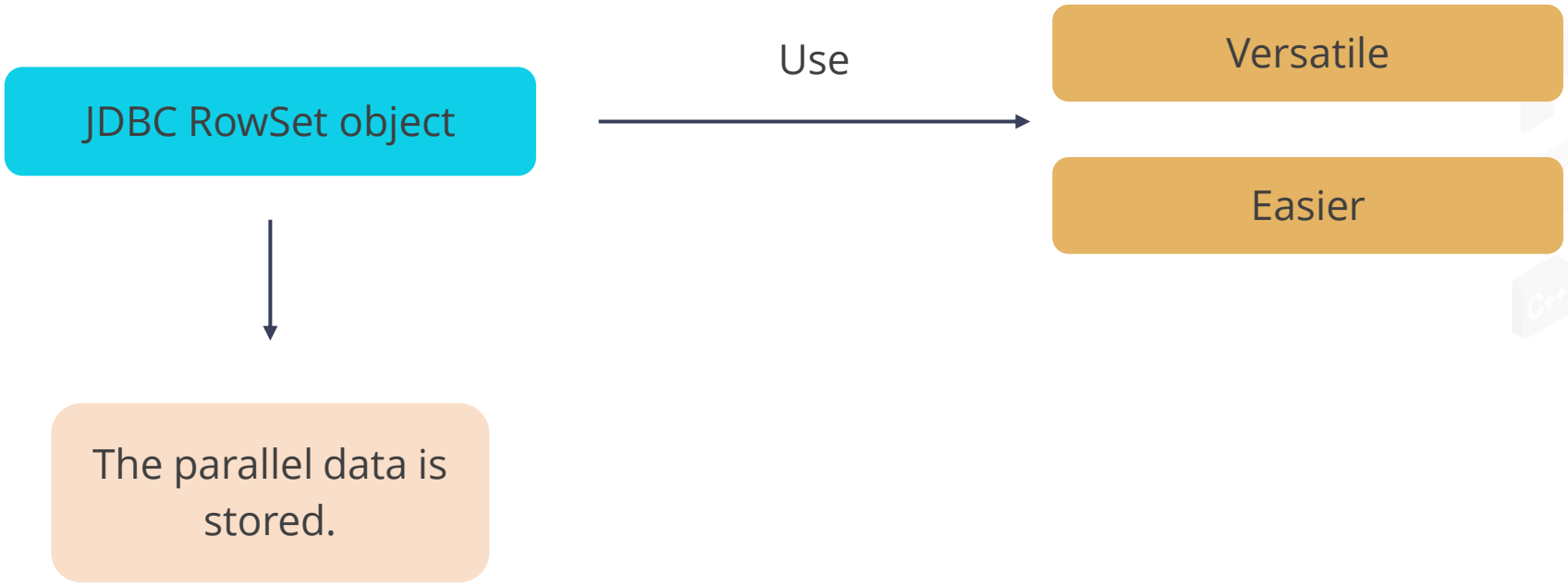
Additional features →

- Scrollability
- Updatability
- Functions as JavaBean's components

©Simplilearn. All rights reserved.

RowSet Objects

Parallel data is stored in the JDBC RowSet object, which makes it more versatile and easier to utilize than the result set.



©Simplilearn. All rights reserved.

RowSet Objects

Oracle has specified five RowSet interfaces for the most common usage, and standard references are available.

Some interfaces that extend these RowSet interfaces are:

Specified →

- JDBCRowSet
- CachedRowSet
- WebRowSet
- JoinRowSet
- FilteredRowSet

©Simplilearn. All rights reserved.

RowSet Objects

All RowSet objects are JavaBeans components, and so they contain:

Properties

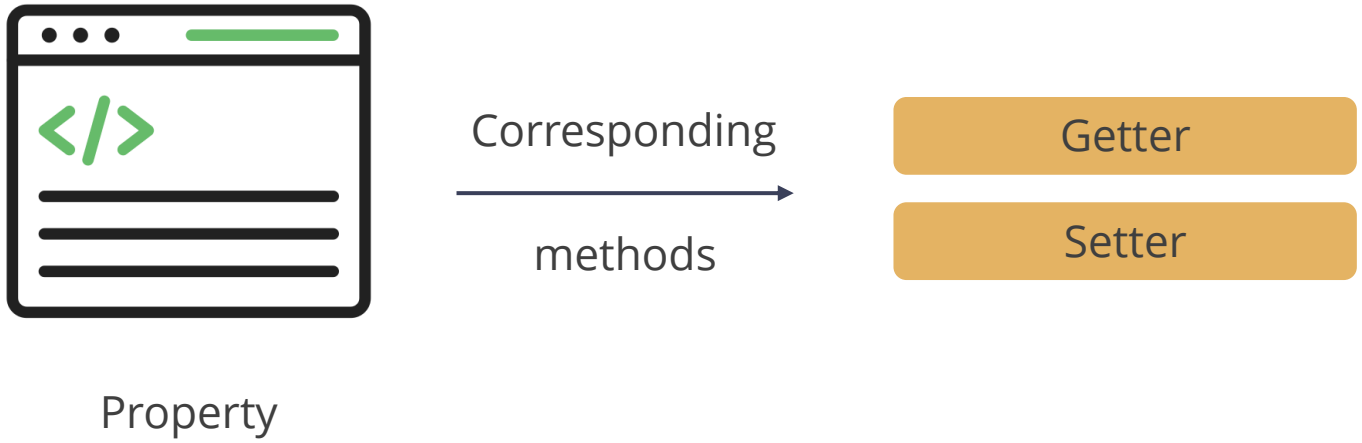
JavaBeans Notification Mechanism

©Simplilearn. All rights reserved.

Properties

Property refers to a field that has the corresponding getter and setter methods.

They are exposed to builder tools that enable visual manipulation of the beans.



©Simplilearn. All rights reserved.

JavaBeans Notification Mechanism

RowSet objects use the JavaBeans event model.
Registered components are notified whenever certain events occur.

Three events trigger notifications for all RowSet objects. They are:

01 The cursor movement

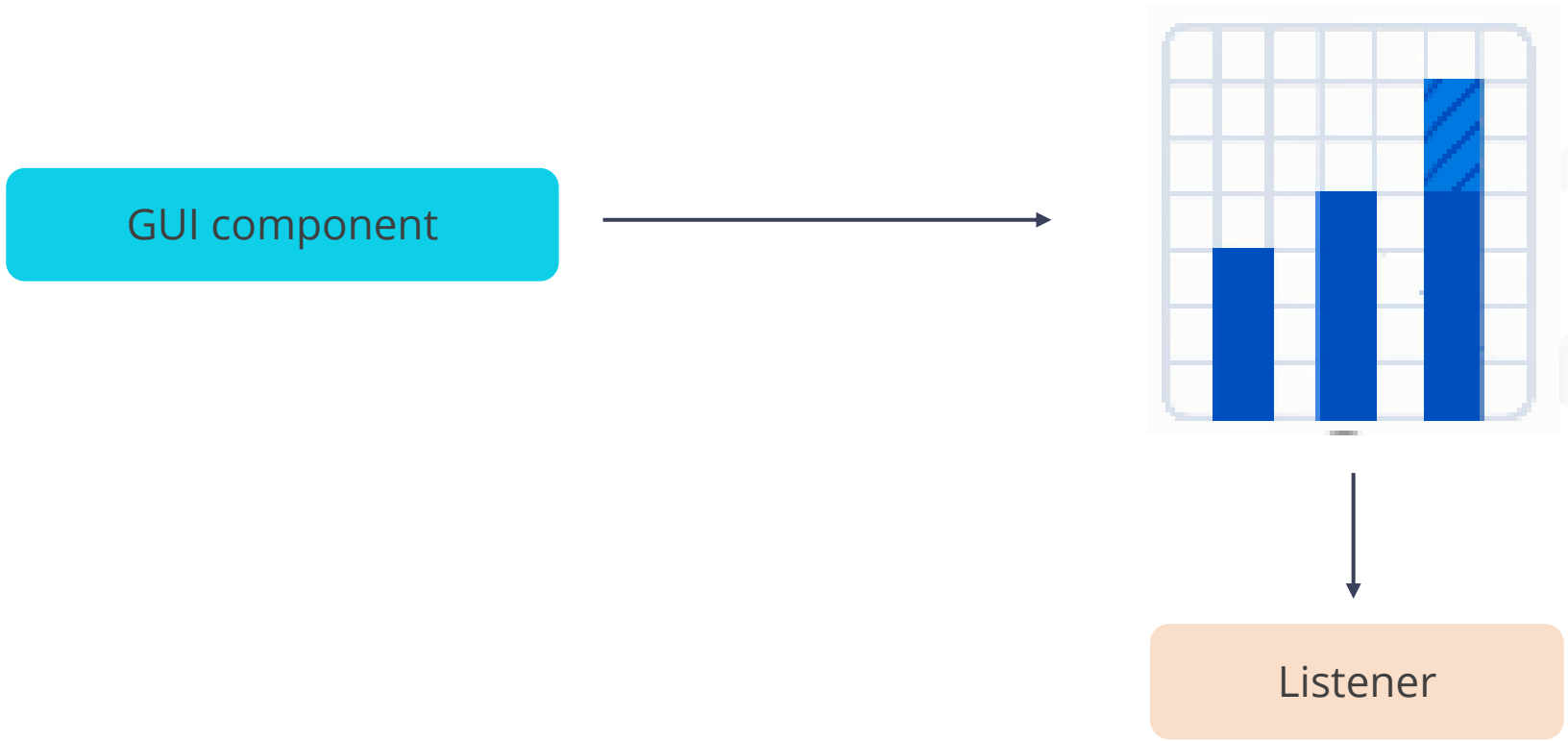
02 Updating, inserting, or deleting the row

03 Changing the entire RowSet

©Simplilearn. All rights reserved.

JavaBeans Notification Mechanism

The event is broadcast to all listeners. Components that have implemented the RowSetListener interface and are added to the component's RowSet objects will receive the event.



Value changes can be noted when the RowSet object is tracked by the bar graph.

©Simplilearn. All rights reserved.

JavaBeans Notification Mechanism

The listener uses RowSetListener methods to define what will happen when a specific event occurs.

A listener should also be added to the list of listeners for RowSet objects.

RowSetListener

Method used

To specify event

©Simplilearn. All rights reserved.

JavaBeans Notification Mechanism

The bar graph component, bg, is registered with the RowSet object, rs, in the code shown here.

```
rs.addListener(bg);
```

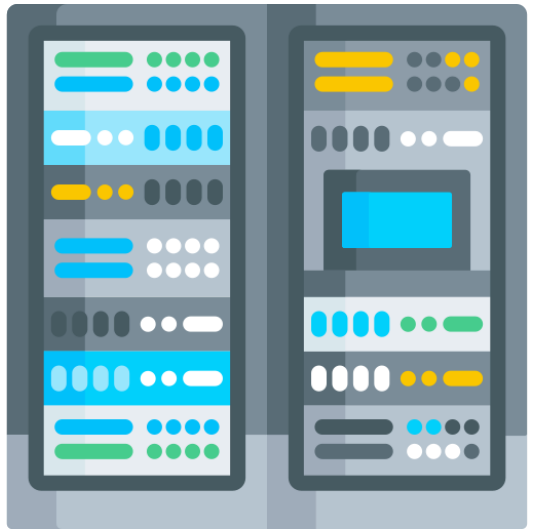
bg will be notified every time the cursor moves or the row changes.

©Simplilearn. All rights reserved.

JavaBeans Notification Mechanism

Some DBMSs do not allow scrollable result sets and do not support updatable result sets.

If the DBMS driver cannot update or scroll through result sets, the RowSet object can be utilized.

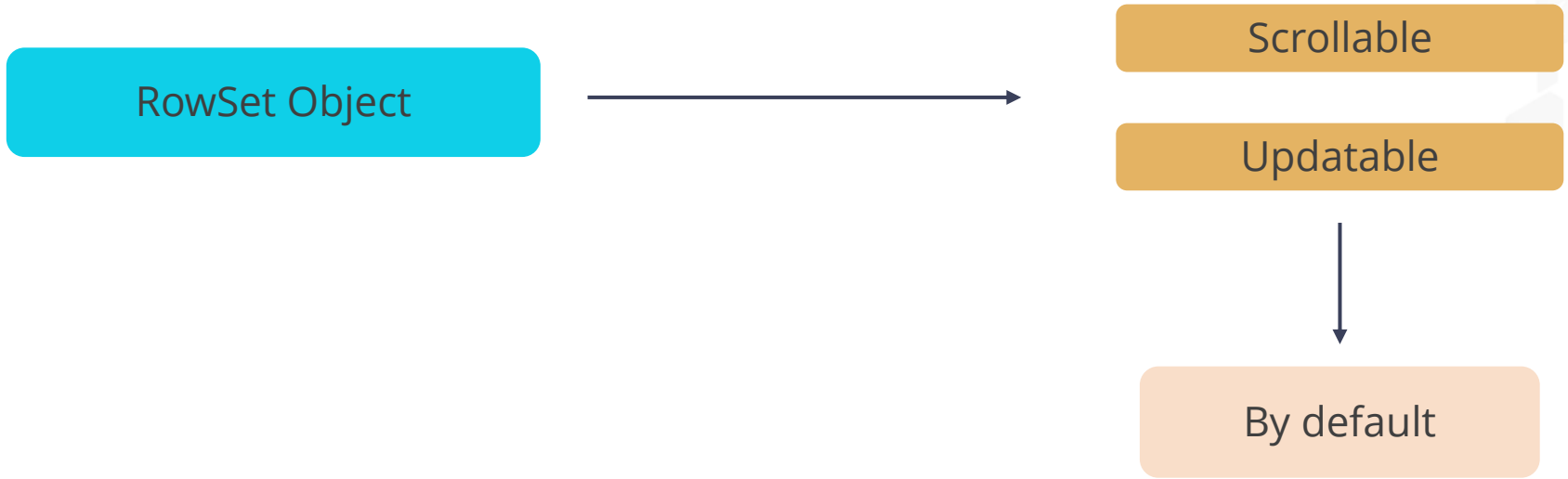


©Simplilearn. All rights reserved.

JavaBeans Notification Mechanism

RowSet object is scrollable and updatable by default.

Users can make the result set scroll and update by populating it with the contents of the result set.



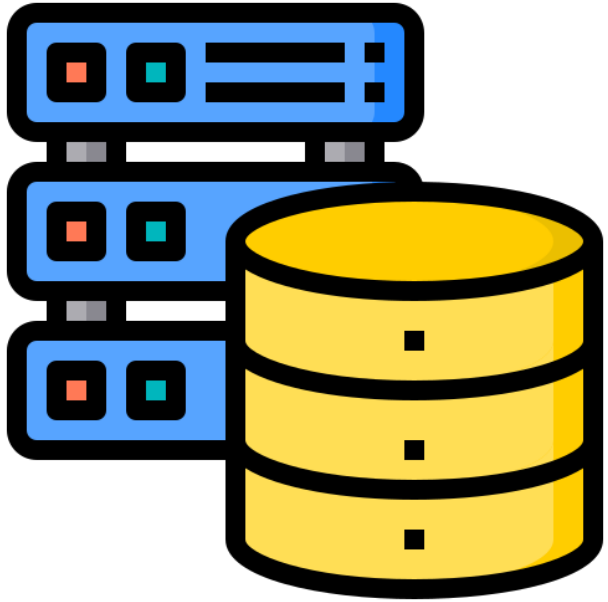
©Simplilearn. All rights reserved.

RowSet Objects Types

©Simplilearn. All rights reserved.

Connected RowSet Objects

The RowSet object has two states: connected and disconnected.

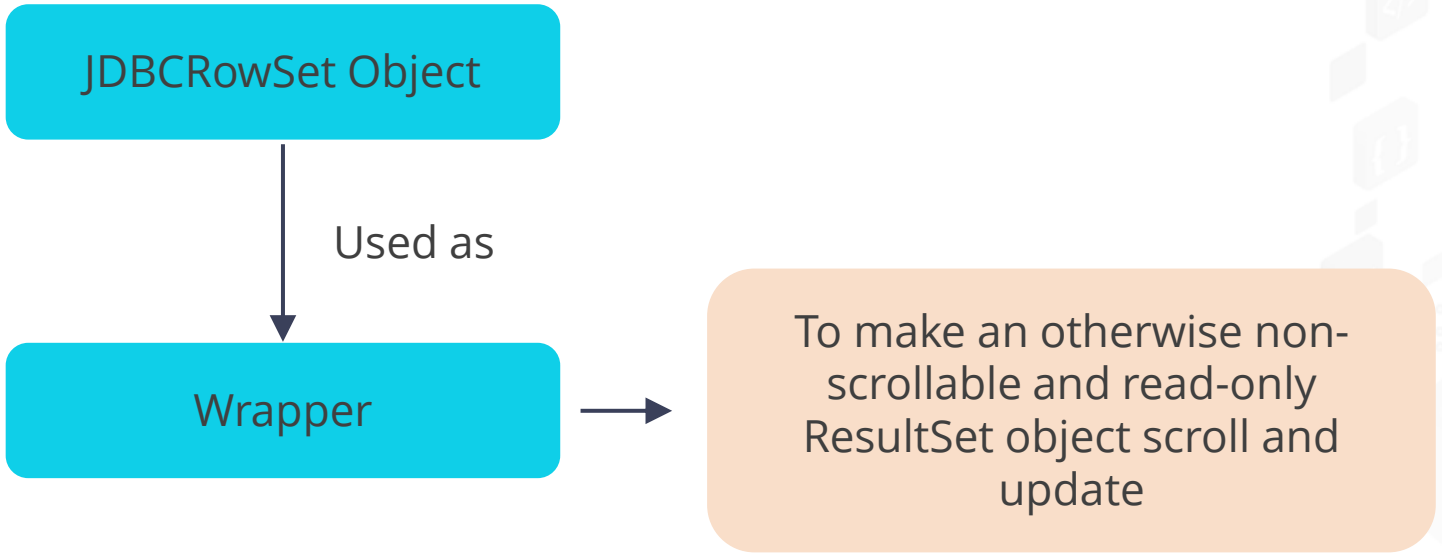


The JDBCRowSet object closely resembles the ResultSet object since it is always linked to the database.

©Simplilearn. All rights reserved.

Connected RowSet Objects

Connected RowSet Objects is used as wrapper.

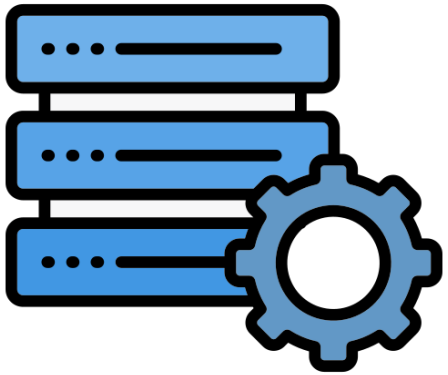


©Simplilearn. All rights reserved.

Disconnected RowSet Objects

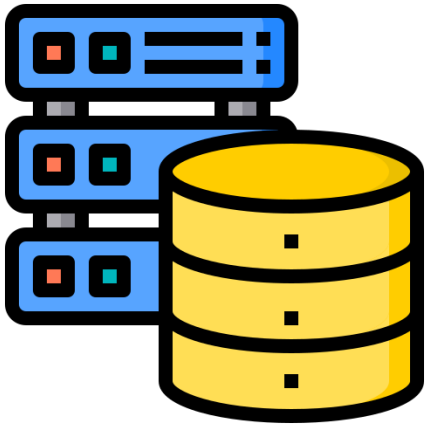
The capabilities of disconnected RowSet objects are the same as those of connected RowSet objects.

These are additional capabilities that are uniquely available to disconnected RowSet objects.



Disconnected RowSet objects

Same



Connected RowSet objects

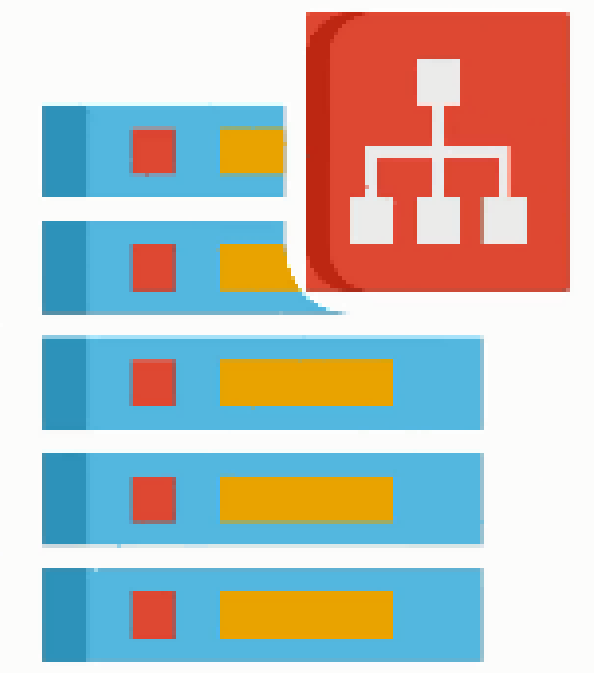
Disconnected RowSet objects are lighter than JDBCRowSet objects or ResultSet objects.

©Simplilearn. All rights reserved.

Disconnected RowSet Objects

RowSet objects that are disconnected can be serialized.

They can also be used to convey information to clients such as PDAs and cell phones.



- CachedRowSet interface
- WebRowSet object
- JoinRowSet object
- FilteredRowSet object

©Simplilearn. All rights reserved.

CachedRowSet

The CachedRowSet interface explains all the essential capabilities of unconnected RowSet objects.

Ways that demonstrate how they are linked:

01 Obtaining the connection to the data source and executing the query

02 Reading the data from the resulting ResultSet object and populating itself with the data

03 Manipulating the data and making changes to it while it is disconnected

04 Reconnecting to the data source to write the changes back to it

05 Checking for conflicts with the data source and resolving those conflicts

©Simplilearn. All rights reserved.

WebRowSet

The WebRowSet object has all the capabilities of a CachedRowSet object and can also perform the following steps:

Writing their own XML document

Reading the XML document which describes the WebRowSet object

©Simplilearn. All rights reserved.

JoinRowSet

The JoinRowSet object has all the capabilities of a WebRowSet object.

It helps to create an equivalent of the SQL JOIN without getting connected to the data source.

JoinRowSet objects

Similar capabilities

WebRowSet objects

©Simplilearn. All rights reserved.

FilteredRowSet

The FilteredRowSet object also has all the capabilities of a WebRowSet object.

Apply filtering criteria to visualize selected data. This is equivalent to executing the query on the RowSet object.

FilteredRowSet objects

Similar capabilities

WebRowSet objects

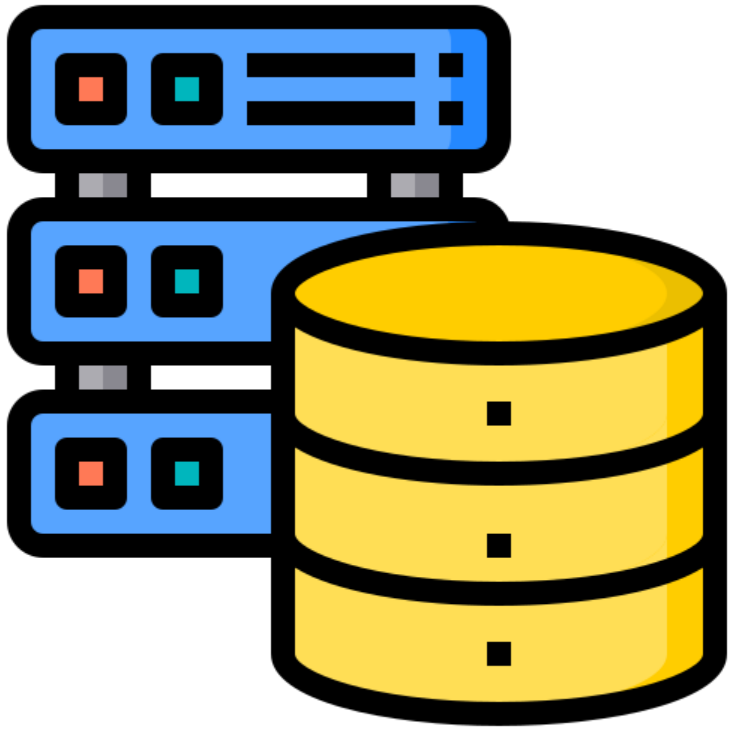
©Simplilearn. All rights reserved.

JDBCRowSet Objects

©Simplilearn. All rights reserved.

JDBCRowSet Objects

The JDBCRowSet object is an upgraded version of the ResultSet object. It provides a more efficient means of retrieving and manipulating data while retaining the connection to the data source.



©Simplilearn. All rights reserved.

JDBCRowSet Objects

The JBDCRowSet object can be created by using an instance of RowSetShop, which is provided by the RowSetProvider class, as shown in the example:

```
RowSetShop  shop = RowSetProvider.newShop();
Try ( JdbcRowSet jdbcRs = shop.createJdbcRowSet()) {
jdbcRs.setUrl(this.settings.urlString);
jdbcRs.setUsername(this.settings.userName);
jdbcRs.setPassword(this.settings.password);
jdbc.Rs.setCommand("select * from COLORS");
jdbc.Rs.execute();
//…….
```

©Simplilearn. All rights reserved.

JDBCRowSet Objects

The RowSetShop interface provides methods to create various types of RowSet implementations, including:

- CachedRowSet
- FilteredRowSet
- JDBCRowSet
- JoinRowSet
- WebRowSet

©Simplilearn. All rights reserved.

JDBCRowSet Objects

Create the JBDCRowSet object with an instance of RowShop. The new JBDCRowSet object will have these properties:

Type:

ResultSet.TYPE_SCROLL_INSENSITIVE (It has the scrollable cursor.)

Concurrency:

ResultSet.CONCUR_UPDATABLE (It can be updated.)

escapeProcessing:

true (When the driver is equipped to handle escape processing, it will scan for any escape syntax and then transform it into a program that the database understands.)

©Simplilearn. All rights reserved.

JDBCRowSet Objects

The new JBDCRowSet object will have these properties:

maxRows:

0 (The number of rows are unlimited in this.)

maxFieldSize:

0 (The number of bytes is unlimited for a column value; it applies only to the columns that store the BINARY, VARBINARY, LONGVARBINARY, CHAR, VARCHAR, and LONG VARCHAR values.)

queryTimeout:

0 (It has no time limit for how long it takes to execute the query.)

©Simplilearn. All rights reserved.

JDBCRowSet Objects

The new JBDCRowSet object will have these properties:

showDeleted:

false (It deletes the rows which are not visible.)

transactionIsolation:

Connection.TRANSACTION_READ_COMMITTED (It reads only the data that has been committed.)

typeMap:

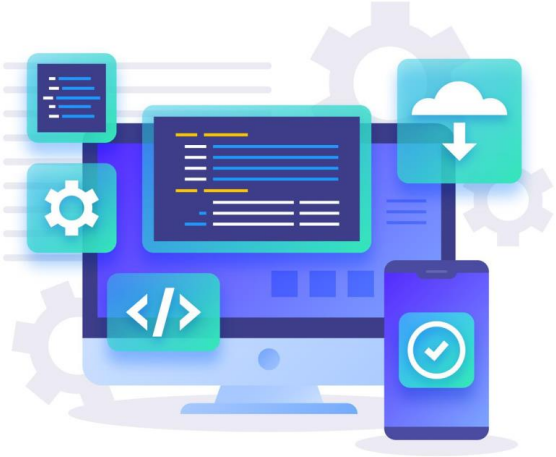
null (The type map associated with the Connection object used by the RowSet object is null.)

©Simplilearn. All rights reserved.

JDBCRowSet Objects

The properties set by default when a new JDBCRowSet object is formed are listed in the default JDBCRowSet object.

Additional properties must be set before populating the JdbcRowSet object with data.



©Simplilearn. All rights reserved.

JDBCRowSet Objects

The JDBCRowSet object must first connect to the database to retrieve its data.

The four properties that contain information used to establish a database connection are:

The Username

The name which the user supplies to the database to gain access

The Password

The user's password that is in the database

The URL

The URL which the user wants to connect to the database

DatasourceName

The name which can be used to retrieve the Datasource object that has been registered with the JNDI naming service

©Simplilearn. All rights reserved.

JDBCRowSet Objects

The command property must be set to specify the SQL query that the JDBCRowSet object will execute.

This property is the query that determines what data the JDBCRowSet object will hold.

©Simplilearn. All rights reserved.

JDBCRowSet Objects: Example

```
jdbcRs.setCommand("select * from COLORS");
```

Command properties refer to the required properties to establish a connection and populate the JBDCRowSet object with data by invoking the execute method.

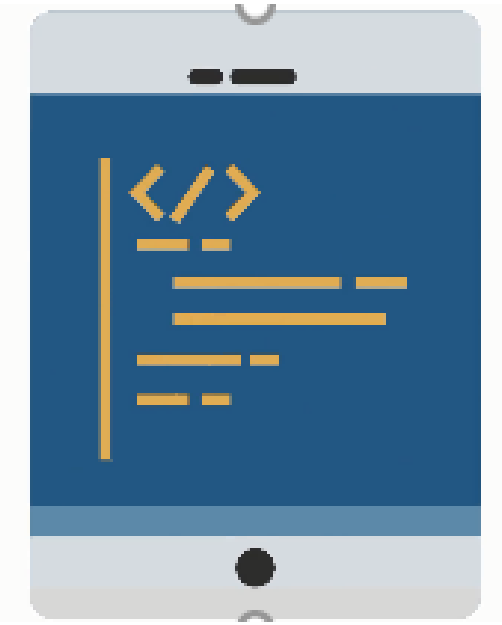
```
jdbcRs.execute();
```

©Simplilearn. All rights reserved.

Execute Method

The execute method helps to perform a variety of tasks in the background.

Data from the ResultSet object is read into the JDBCRowSet object.

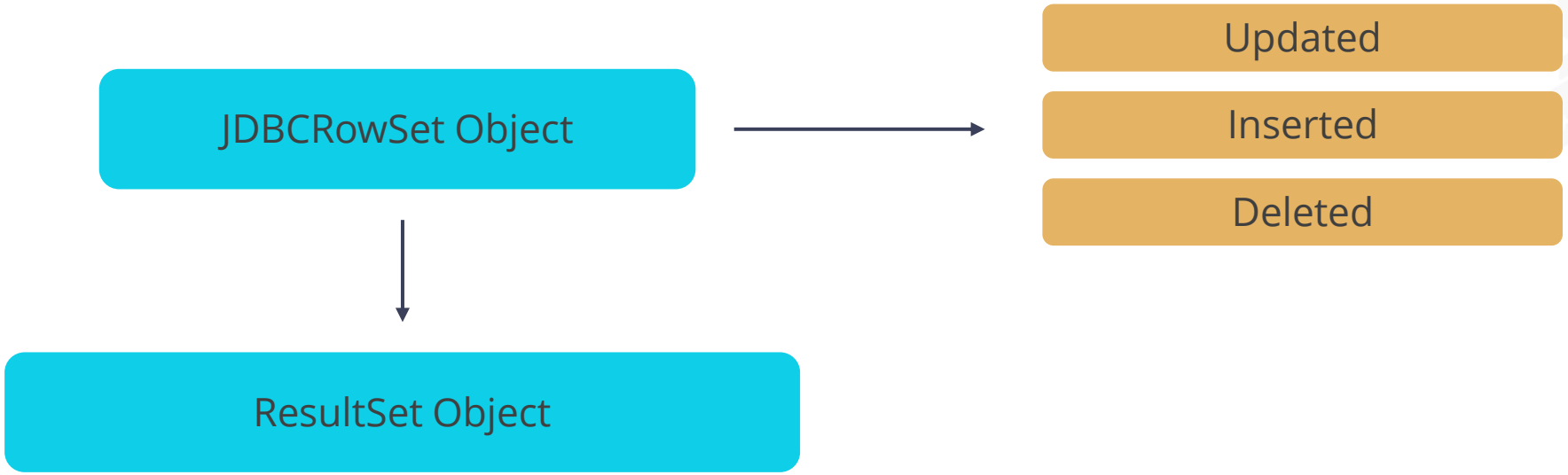


©Simplilearn. All rights reserved.

JDBCRowSet Object

Rows in the JDBCRowSet object can be updated, inserted, and deleted the same way you update, insert, and delete rows in the ResultSet object.

Explore the scrollable ResultSet object and navigate the JDBCRowSet object



©Simplilearn. All rights reserved.

JDBCRowSet Object

By default, it is both scrollable and updatable, maintaining a persistent connection to the database for real-time data manipulation.

JDBCRowSet Object → Identical → ResultSet Object

↓

Dealing with it is equivalent to working with the ResultSet object.

©Simplilearn. All rights reserved.

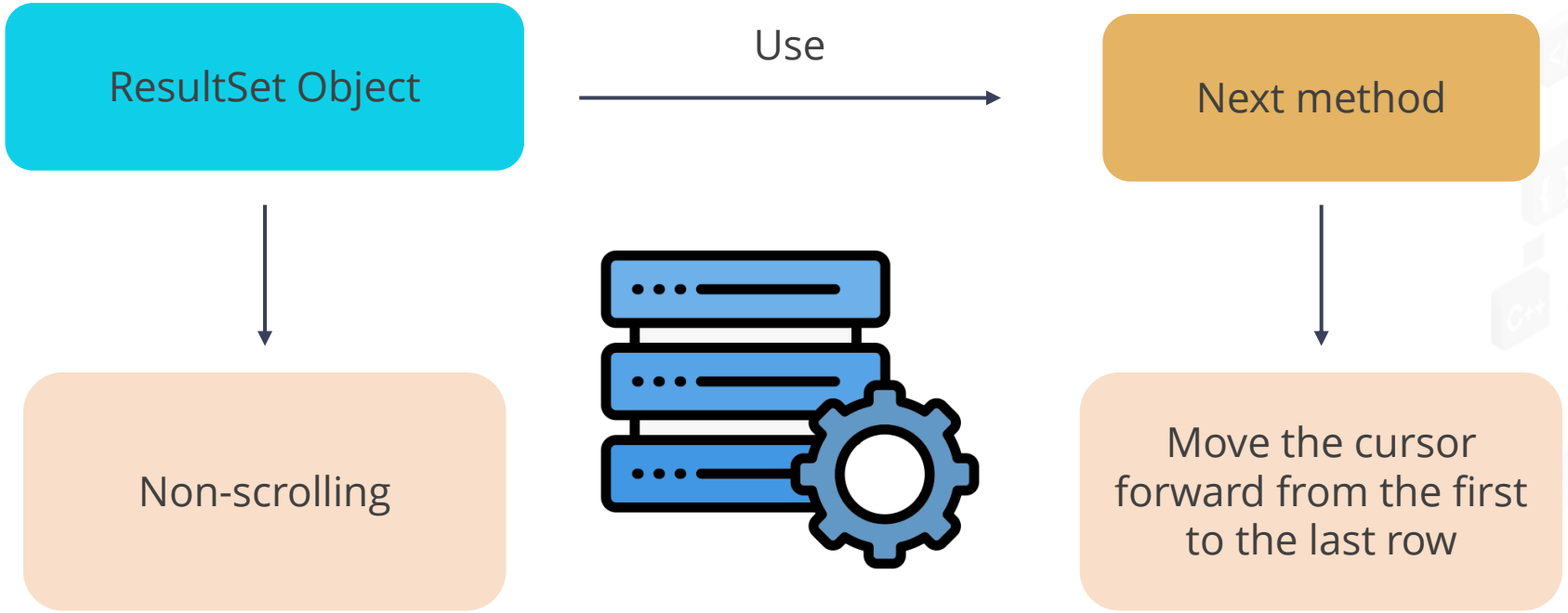
simplilearn

Navigate JDBCRowSet Objects

©Simplilearn. All rights reserved.

JDBCRowSet Objects

A non-scrolling ResultSet object can only use the Next method to move its cursor forward.



©Simplilearn. All rights reserved.

JDBCRowSet Objects

All the cursor movement methods described in the ResultSet interface are available by default.

It usually calls the JDBCRowSet object, as shown in the example here:

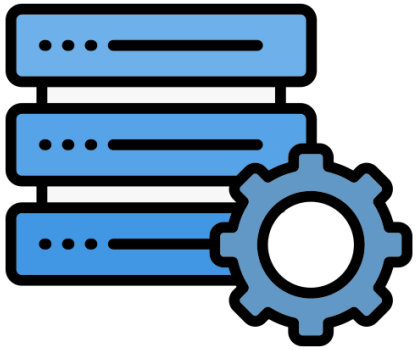
```
jdbcRs.absolute(3);
jdbcRs.previous();
```

©Simplilearn. All rights reserved.

JDBCRowSet Objects

In the JDBCRowSet object, the first method is like the next function.

Move the pointer to the place after the last row



Move the cursor to the beginning of ResultSet

©Simplilearn. All rights reserved.

Updating the Column Values

Update the data in the JDBCRowSet object in the same way the data in the ResultSet object was updated

©Simplilearn. All rights reserved.

Updating the Column Values: Example

A fruit seller wishes to raise the price of a peach. If the seller knows that the peach is in the third row of the jdbcRs object, the code can be written as:

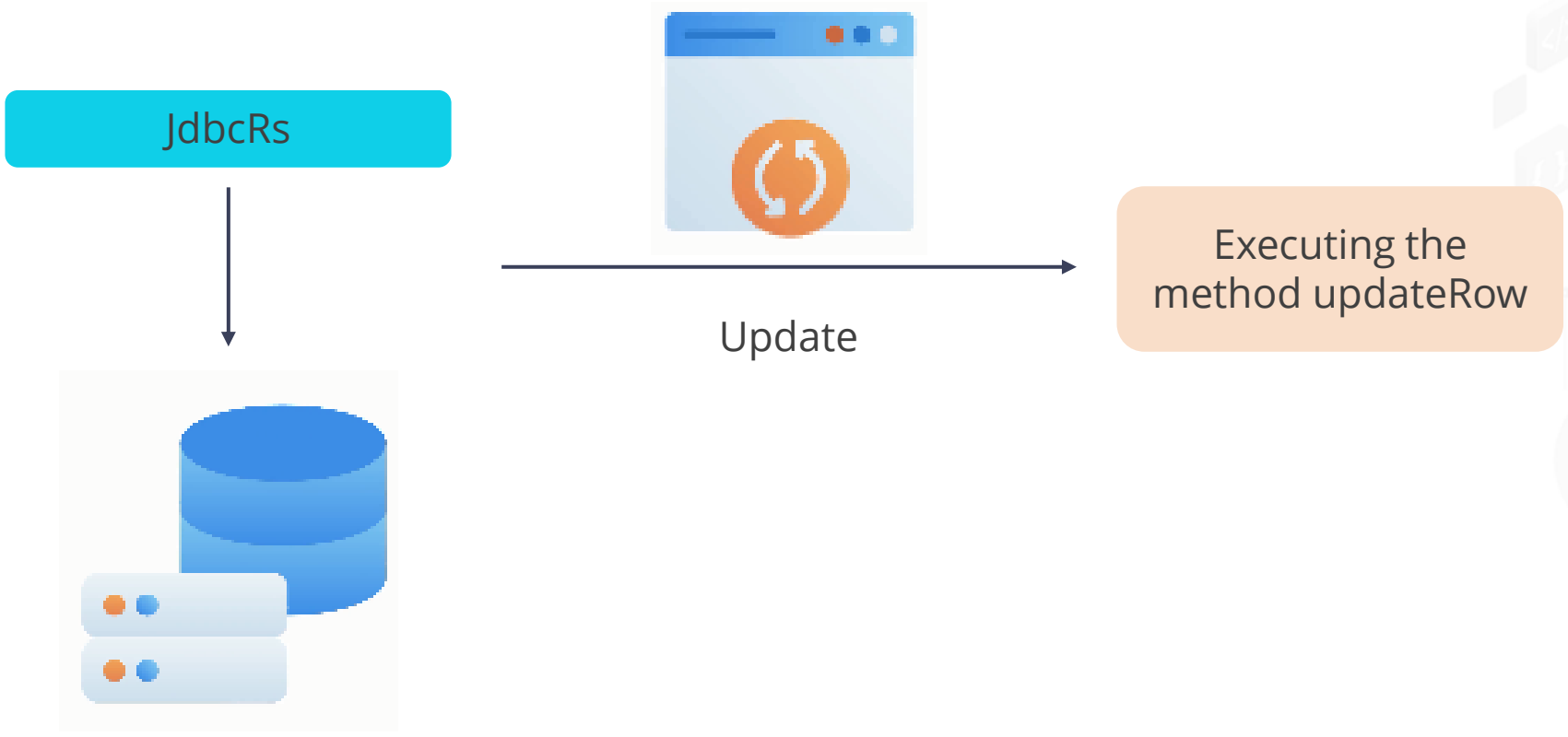
```
jdbcRs.absolute(2);
jdbc.updateFloat("price", 500);
jdbc.updateRow();
```

The code moves the cursor to the third row and updates the database with the new price by changing the value of the column PRICE to 200.

©Simplilearn. All rights reserved.

Updating the Column Values: Example

JDBCRs has maintained its connection to the database and can update it by executing the updateRow method.

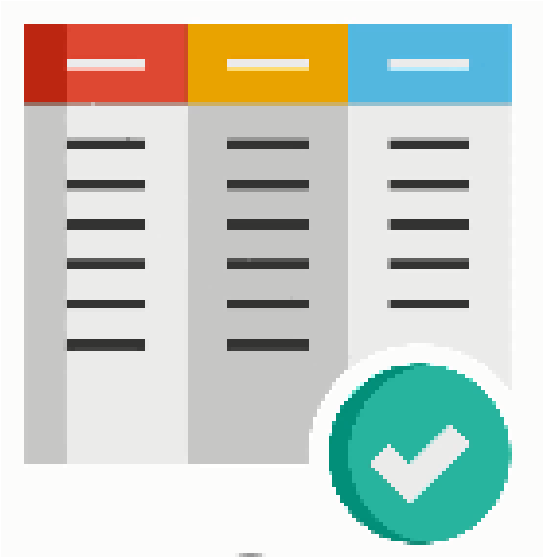


©Simplilearn. All rights reserved.

Insert Rows

The fruit shop owner wants to add one or more fruits to their store.

The owner must add one row to the fruits table for each new fruit, as shown in the JDBCRowSet sample code fragment.

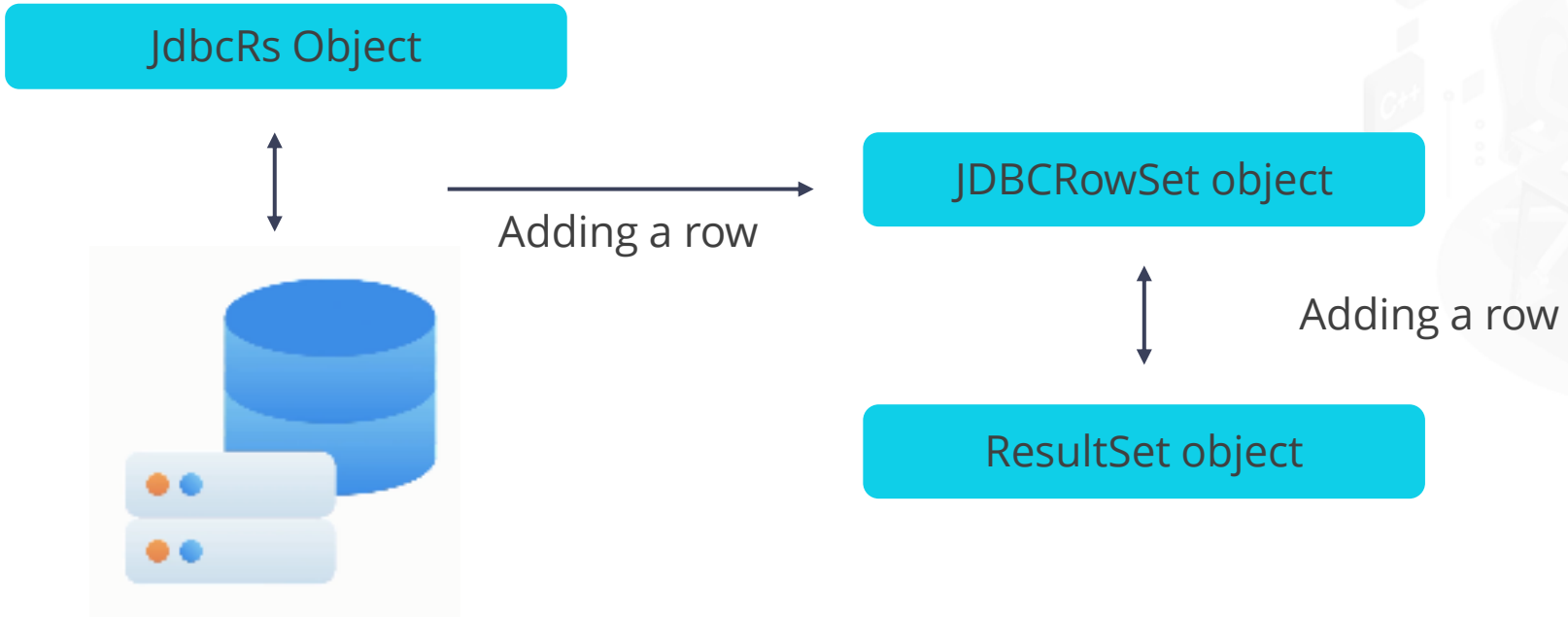


©Simplilearn. All rights reserved.

Insert Rows

The JdbcRs object is always connected to the database. Adding a row to the JDBCRowSet object is the same as adding a row to a ResultSet object.

To insert the row, move the cursor, use the relevant updater method to set the value for each column, and execute the insertRow method.



©Simplilearn. All rights reserved.

Insert Rows

The code for inserting a row is as shown below:

```
jdbcRs.moveToInsertRow();
jdbcRs.updateString("FRUIT_NAME", Orange");
jdbcRs.updateInt("ID", 24);
jdbcRs.updateFloat("PRICE", 100);
jdbcRs.updateInt("SALES",0);
jdbcRs.updateInt("TOTAL", 0);
jdbcRs.insertRow();
jdbcRs.moveToInsertRow();
jdbcRs.updateString("FRUIT_NAME", "Grapes");
jdbcRs.updateInt("ID", 24);
jdbcRs.updateFloat("PRICE", 250);
jdbcRs.updateInt("SALES", 0);
jdbcRs.updateInt("TOTAL", 0);
jdbcRs.insertRow();
```

©Simplilearn. All rights reserved.

Insert Rows

On using the insertRow method, a new row will be generated both in the jdbcRs object and the database.

The method is repeated twice in the above code, resulting in two new rows being entered into the jdbcRs object and the database.

insertRow method → A new row is created in the jdbcRs object.

©Simplilearn. All rights reserved.

Insert Rows

The JDBCRowSet and ResultSet objects have the same functionality for updating and inserting new row data and deleting rows.

To delete a row, use the below syntax:

```
jdbcRs.last();
jdbcRs.deleteRow();
```

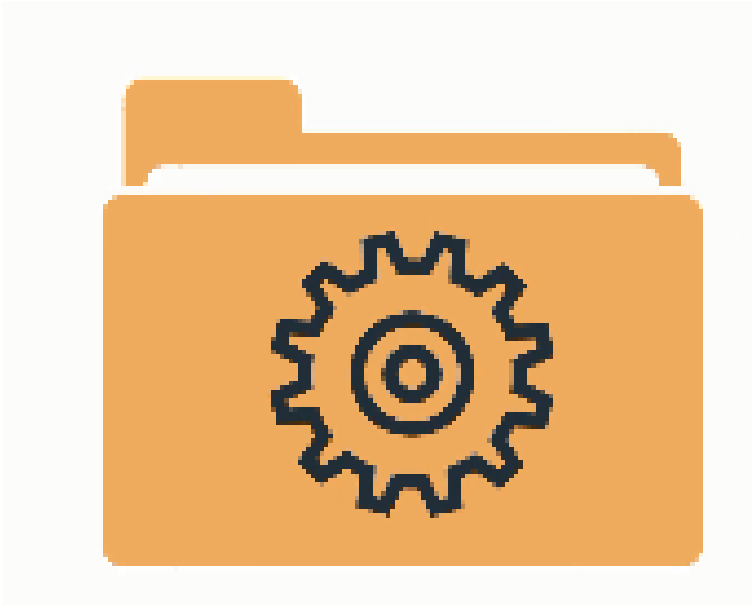
©Simplilearn. All rights reserved.

CachedRowSet Objects

©Simplilearn. All rights reserved.

CachedRowSet Objects

The CachedRowSet object doesn't need to be connected to its data source to function.



The RowSet object is not connected to the database and gets its name because it is stored in its data memory.

©Simplilearn. All rights reserved.

CachedRowSet Objects

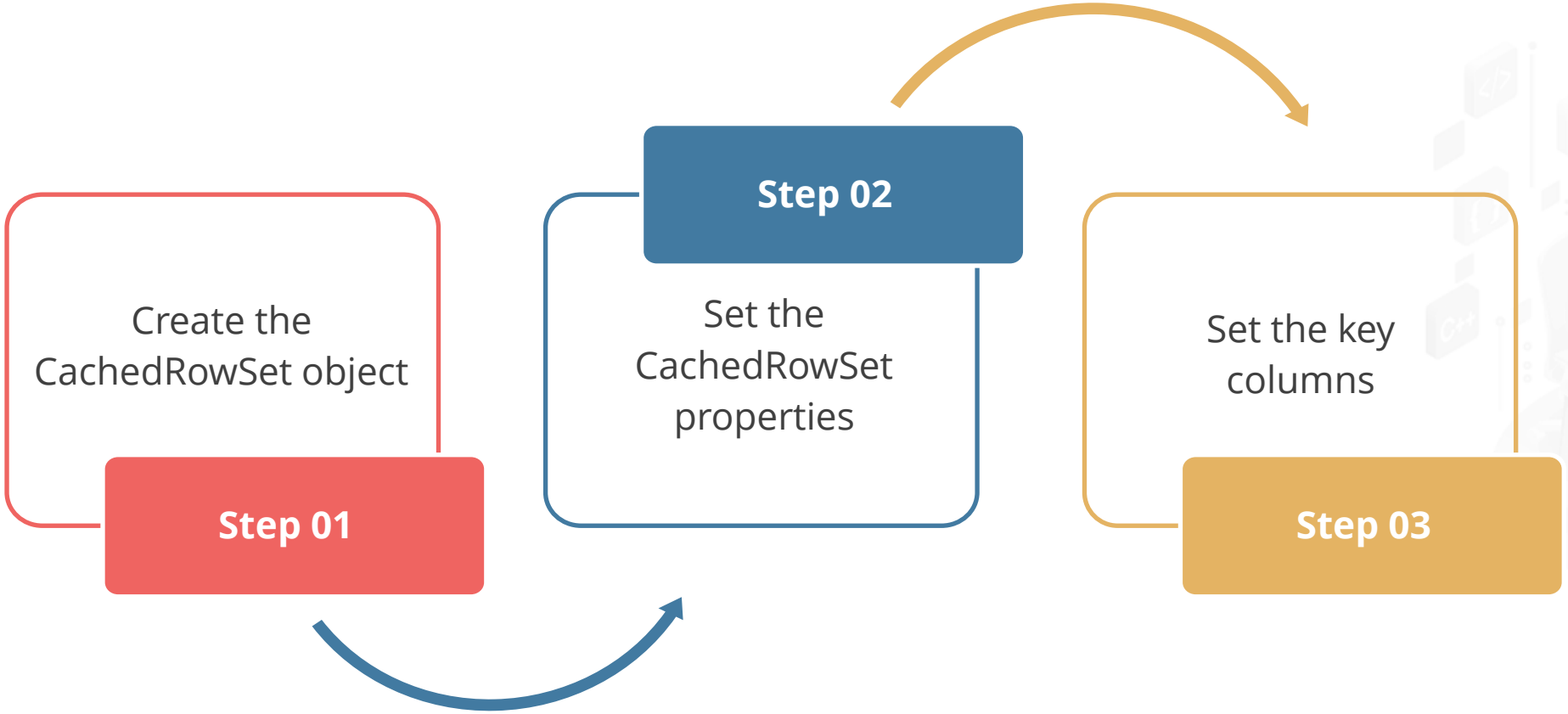
The CachedRowSet interface is a superinterface for all unconnected RowSet objects; everything shown here applies to the:

- WebRowSet objects
- JoinRowSet objects
- FilteredRowSet objects

©Simplilearn. All rights reserved.

CachedRowSet Objects

The steps to set up CachedRowSet Objects are:



©Simplilearn. All rights reserved.

Create CachedRowSet Objects

It uses an instance of RowSetShop, generated from the RowSetProvider class, to create the new CachedRowSet object.

Below is an example where the CachedRowSet object is created from the CachedRowSetSample:

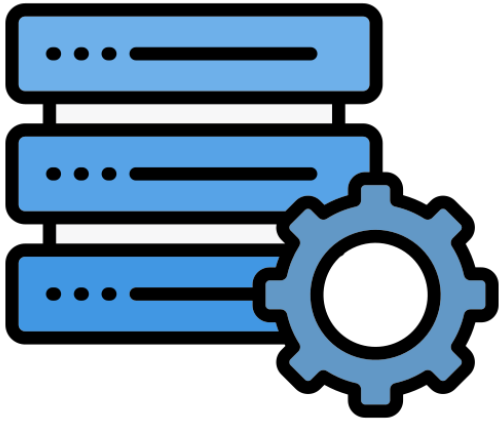
```
RowShop shop = RowSetProvider.newShop();
CachedRowSet crss = shop.createCachedRowSet();
```

©Simplilearn. All rights reserved.

Create CachedRowSet Objects

Default settings for the properties of the **crss** object are the same as those of the JDBCRowSet object, which is declared first.

It is an instance of RIOptimisticProvider, which is the default SyncProvider implementation.



©Simplilearn. All rights reserved.

Create CachedRowSet Objects

The SyncProvider object offers a RowSetReader for reading data and a RowSetWriter for writing data.

Both are separated from the RowSet object to fetch data back to its source.

Source

Create

Update

©Simplilearn. All rights reserved.

Set the CachedRowSet Properties

Although the default values of properties are usually sufficient, the setter method allows one to modify the value of a property as needed.

There are a couple of properties that have no default settings.

Disconnected RowSet objects

It should be able to connect to the data source and pick the required data.

©Simplilearn. All rights reserved.

Set the CachedRowSet Properties

The specified properties contain the information required to establish a database connection:

Username

The name the user supplies to the database to gain access

Password

The SQL database uses the password for it.

URL

The user wants to connect their Java application through the JDBC URL of the database.

DatasourceName

The name used to retrieve the DataSource object that has been registered with the JNDI naming service.

©Simplilearn. All rights reserved.

Set the CachedRowSet Properties

The code sets the username, password, and URL attributes, allowing the Driver Manager class to obtain the connection.

Syntax:

```
Public static void setConnectionProperties(
String username, String pwd )  {
crss.setUsername ( username );
crss.setPassword ( pwd );
crss.setUrl( "jdbc:mySubprotocol:mySubname");
```

©Simplilearn. All rights reserved.

Set the CachedRowSet Properties

The property that needs to be set is the command property.

The data is read into the RowSet object from the ResultSet object. Query which produces that ResultSet object is the value for the command property.

©Simplilearn. All rights reserved.

Set the CachedRowSet Properties: Example

The code sets the command property with the query that produces the ResultSet object containing all the data in the COLORS table.

```
crss.setCommand( "select * from COLORS" );
```

©Simplilearn. All rights reserved.

Setting Key Columns

To make any changes to the **crs** object and save it on the database, set one or more bits of information, known as key columns.

Key columns, like primary keys, are important because they indicate one or more columns that uniquely identify a row.

The difference is that the main key is put on the database table, while the key columns are set on the specific RowSet object.

©Simplilearn. All rights reserved.

Setting Key Columns

The code shown below moves the crss key columns to the second column:

```
int [] keys = { 2 };
crss.setKeyColumns(keys);
```

©Simplilearn. All rights reserved.

Setting Key Columns

The item ID column is the first column in the COLORS table and serves as the key column. It uniquely identifies only one row in the table COLORS.

In the COLORS table's definition, this column is also designated as the main key.

"setKeyColumns()"

It takes an array as input to allow it to take one or more columns to differentiate rows.

©Simplilearn. All rights reserved.

How to Populate CachedRowSet Objects

Populating a disconnected RowSet object requires more effort than populating a connected RowSet object.

©Simplilearn. All rights reserved.

How to Populate CachedRowSet Objects

The following syntax populates the crss object after setting up the CachedRowSet object:

```
crss.execute();
```

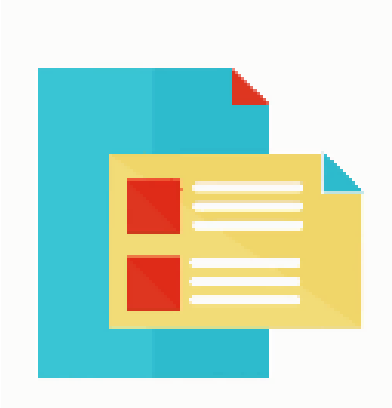
The CRSS data contains the ResultSet object, which is generated by executing the execute() method.

©Simplilearn. All rights reserved.

How to Populate CachedRowSet Objects

The CachedRowSet implementation contains more methods and implementations than the JDBC Row Set.

CachedRowSet



Method

The detached RowSet object reader will work behind the scenes to populate the RowSet object with data when an application uses the execute method.

©Simplilearn. All rights reserved.

simplilearn

How to Populate CachedRowSet Objects

The newly generated CachedRowSet object is not connected to the data source and needs to be connected to access data.

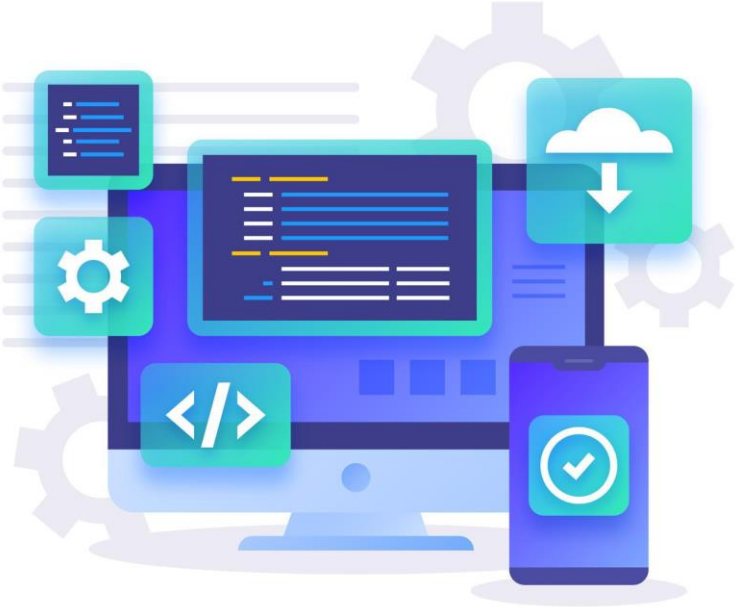
The Default SyncProvider object provides a reader, which establishes a connection using:

- Username
- Password
- Url
- datasourceName

©Simplilearn. All rights reserved.

How to Populate CachedRowSet Objects

The reader that performs the command's query reads the data from the ResultSet object created by the query and populates the CachedRowSet object with it.



©Simplilearn. All rights reserved.

Operations on CachedRowSet Objects

©Simplilearn. All rights reserved.

How to Update Column Values

Data in the CachedRowSet object can be updated in the same way as the JDBCRowSet object.

For example, in a row whose item ID column includes an item identification of 7501, code from CachedRowSetSample.java helps to increment the value in the column QUANTITY by 1:

```
While (crss.next()) {
System.out.println( "Found item" + crss.getInt("item_ID") + ":"  +
crs.getString("item_NAME") );
If (crs.getInt("item_ID") == 7501 ) {
Int currentQuantity = crss.getInt("QUANTITY") + 1;
System.out.println(" Updating quantity to " + currentQuantity);
crss.updateInt( " QUANTITY ", currentQuantity  + 1 );
crss.updateRow();
crs.acceptChanges( con );
}
}
```

©Simplilearn. All rights reserved.

How to Update Column Values

In the CachedRowSet object, the code for updating the column value and inserting and deleting rows is the same as in the JDBCRowSet object.

The CachedRowSample.java extract shown here adds a new row to the CachedRowSet object crss:

```
crss.moveToInsertRow();
crss.updateInt("item_ID", newItemId );
crss.updateString("item_NAME", "Tech34");
crss.updateInt("ID", 9767);
crss.update("QUANTITY", 14);
Calendar timeStamp;
timeStamp = new GregorianCalendar();
timeStamp.set(4000, 3, 2);
crss.updateTimestamp( "DATE_VAL", new
Timestamp(timeStamp.getTimeInMillis()));
crss.insertRow();
crss.moveToCurrentRow();
```

©Simplilearn. All rights reserved.

How to Locate and Delete Rows

This example locates and deletes the row with the value 4421 in the item ID column from the CachedRowSet crss:

```
While (crss.next()) {
If (crs.getInt("item_ID") == 4421 ) {
crs.deleteRow();
break;
}
}
```

©Simplilearn. All rights reserved.

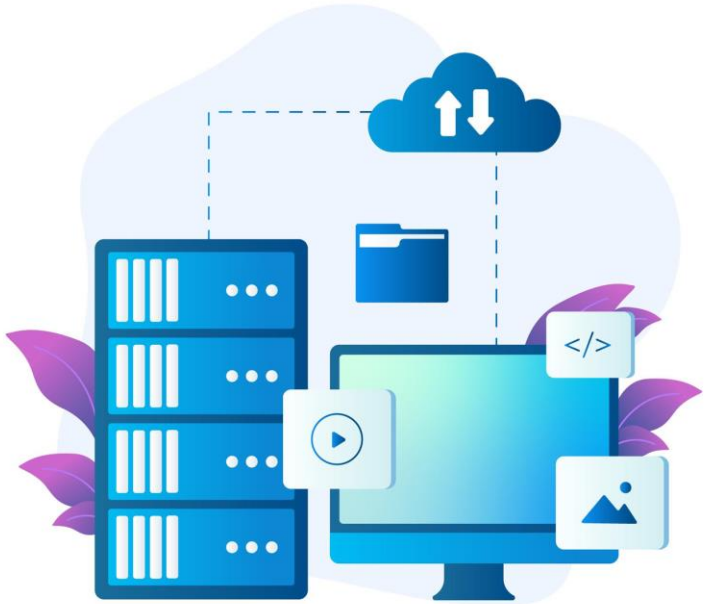
How to Update Data Sources

The JDBCRowSet object is connected to its data source, and the methods updateRow, insertRow, and delete on the JDBCRowSet object differ significantly from those on the CachedRowSet object.

©Simplilearn. All rights reserved.

How to Update Data Sources

While methods can modify the data stored in a CachedRowSet object's memory, they cannot alter the data source for disconnected RowSet objects.



©Simplilearn. All rights reserved.

How to Update Data Sources

Modifications to the data source and disconnected RowSet object must use the method acceptChanges.

Syntax:

```
crss.acceptChanges();
```

The method acceptChanges, like the method execute, does its work invisibly.

The method acceptChanges assigns its tasks to the RowSet object's writer, while the method execute assigns its work to the RowSet object's reader.

©Simplilearn. All rights reserved.

How to Update Data Sources

The writer opens the database connection in the background and updates the database with the modifications made to the RowSet object before closing the connection.

A conflict may occur if the other party has updated a value in the database that matches the updated value in the RowSet object.

©Simplilearn. All rights reserved.

How to Update Data Sources

Which value should now be stored in the database?

01

The RIXMLProvider implementation used by the WebRowSet object is an example of this.

02

The writer, on the other hand, ensures that no conflicts arise by enforcing database locks that prevent others from making modifications.

03

The crss object's writer is RIOptimisticProvider, which is provided by the default SyncProvider implementation.

©Simplilearn. All rights reserved.

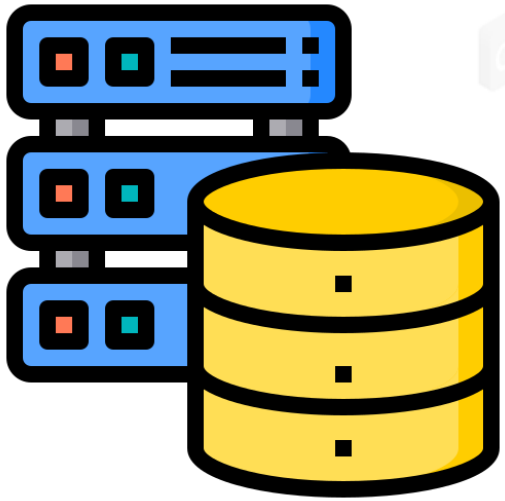
How to Update Data Sources

The RIOptimisticProvider implementation derives its name from using an optimistic concurrency model, which aims to minimize conflicts and eliminate the requirement for database locks.

In the absence of conflicts, the writer propagates any modifications made to the crss object to the database, making the changes permanent.

RIOptimisticProvider

Implementation →



©Simplilearn. All rights reserved.

How to Update Data Sources

When the writer detects one or more conflicts, it creates a SyncResolver object that holds the database values responsible for causing them.

The method acceptChanges throws a SyncProviderException object, which an application can catch and use to retrieve the SyncResolver object.

©Simplilearn. All rights reserved.

How to Update Data Sources

The syntax shown below retrieves the object resolver SyncResolver:

```
Try {
crss.acceptChanges();
} catch (SyncProviderException spex) {
SyncResolver resolver = spex.getSyncResolver();
}
```

©Simplilearn. All rights reserved.

How to Update Data Sources

01

Object resolver is the RowSet object which replicates the crss object except that it contains only the values in the database that cause the conflict, and all other column values are null.

02

Loop through the resolver object's rows to find the values that aren't null and values that cause conflict

03

Find the value in the crss object at the same location and compare them

©Simplilearn. All rights reserved.

How to Update Data Sources

The code segment fetches the resolver and iterates through the rows with conflicting values using the SyncResolver function nextConflict.

The resolver object gets the status of each conflicting value and updates the value by UPDATE ROW CONFLICT.

The function then looks for the column in that row of the resolver object with the conflicting value, which will be a non-null value.

The code now transfers the crss object's cursor to the same row.

©Simplilearn. All rights reserved.

How to Update Data Sources

The function setResolvedValue updates the value in both the crss object and the database as demonstrated in the CachedRowSample code below:

```
Try {
System.out.println( " add  new…. " );
crss.acceptChanges(con);
System.out.println( " added the row ….” );
this.viewTable( con );
} catch ( SyncProviderException spe ) {
SyncResolver resolver = spe.getSyncResolver();
Object crssValue;
Object resolverValue;
Object resolvedValue; // value to be persisted
While ( resolver.nextConflict() ) {
If (resolver.getStatus() = SyncResolver.INSERT_ROW_CONFLICT) {
Int row = resolver.getRow();
crss.absolute( row );
```

©Simplilearn. All rights reserved.

How to Update Data Sources

```
Int colCount = crs.getMetData().getColumnCount();
For (int j = 1; j <= colCount; j++) {
If (resolver .getConflictValue(j) != null ) {
crssValue = crs.getObject(j);
resolverValue = resolver.getConflictValue(j);
resolvedValue = crssValue;
resolver.setResolvedValue( j, resolvedValue );
}
}
}
}
}
```

©Simplilearn. All rights reserved.

Listeners

©Simplilearn. All rights reserved.

RowSet Object

The RowSet object informs other components when specific events occur.

If the data in the RowSet object changes, the RowSet object notifies those who are interested.

One significant aspect of this notification technique is that users must either add or remove the components that will be notified as an application programmer.

©Simplilearn. All rights reserved.

How to Set up Listeners

RowSet object's listener implements a few methods from the RowSetListener interface.

Method	Use
cursorMoved	Defines what the listener should do when the cursor moves in the RowSet object
rowChanged	Defines what the listener should do if one or more column values in the row change
rowSetChanged	Defines what the listener should do after the RowSet object has been populated with new data or information

©Simplilearn. All rights reserved.

BarGraph Object

The BarGraph object visualizes the data from the RowSet object.

The BarGraph Object can update as the data changes to reflect the new data.

©Simplilearn. All rights reserved.

crss Objects

This example indicates that when the cursor for the crss objects moves, the values in crss are updated.

```
crss.addRowSetListener( bar );
```

Users can also stop the notifications by removing the listener using the following code:

```
crss.removeRowSetListener(bar);
```

©Simplilearn. All rights reserved.

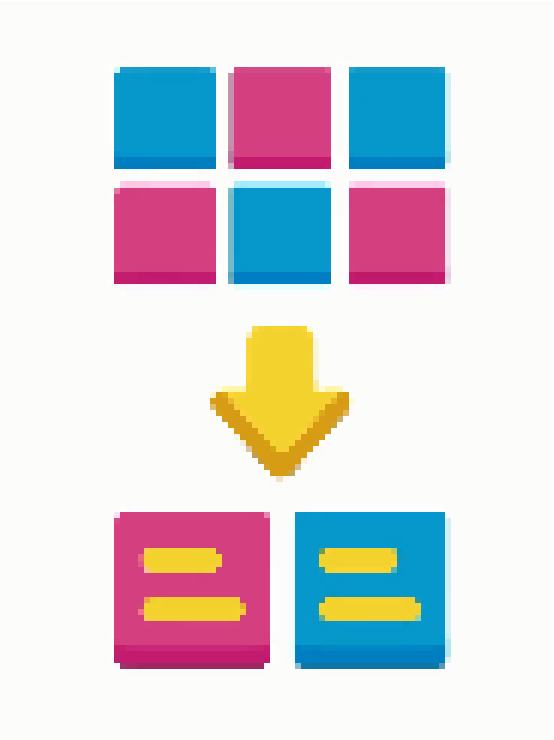
FilteredRowSet Objects

©Simplilearn. All rights reserved.

FilteredRowSet Object

The FilteredRowSet object reduces the number of rows visible in the RowSet object.

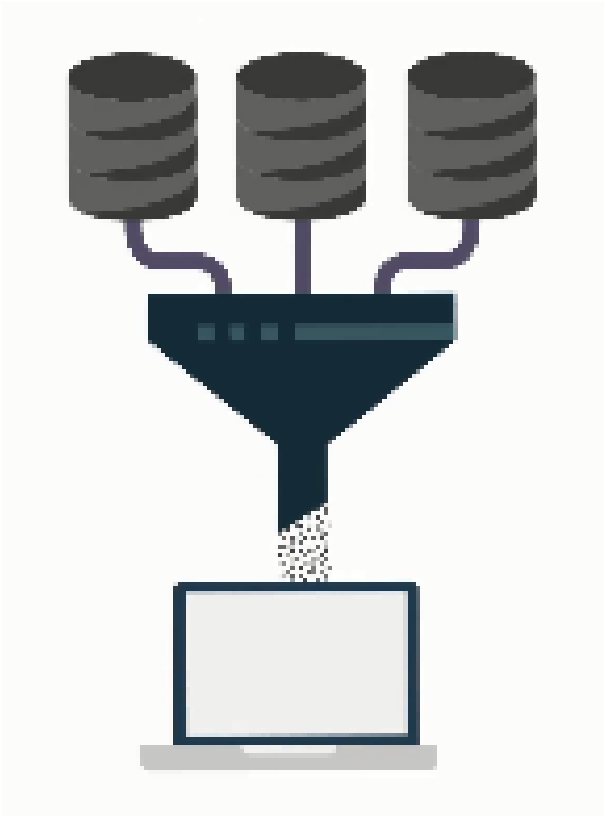
It allows the user to deal with data relevant to the current task.



©Simplilearn. All rights reserved.

FilteredRowSet Object

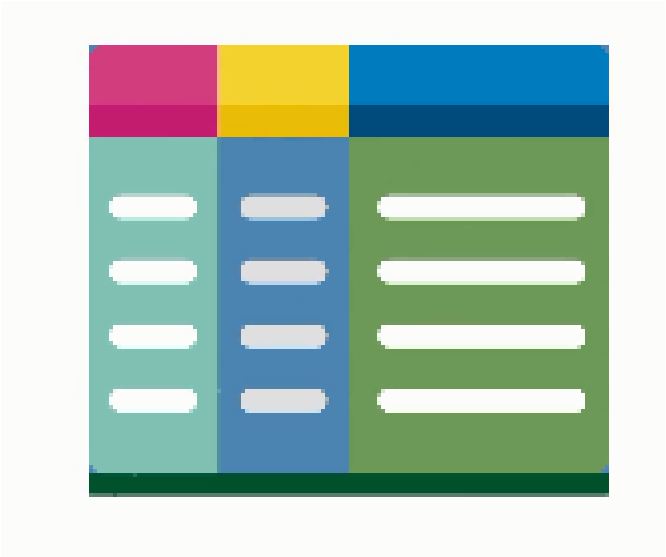
FilteredRowSet objects display only the rows of data that fall within the parameters specified by the user.



©Simplilearn. All rights reserved.

FilteredRowSet Object

By executing a query to the data source, the JDBCRowSet object can perform filtering and display only the desired columns and rows.

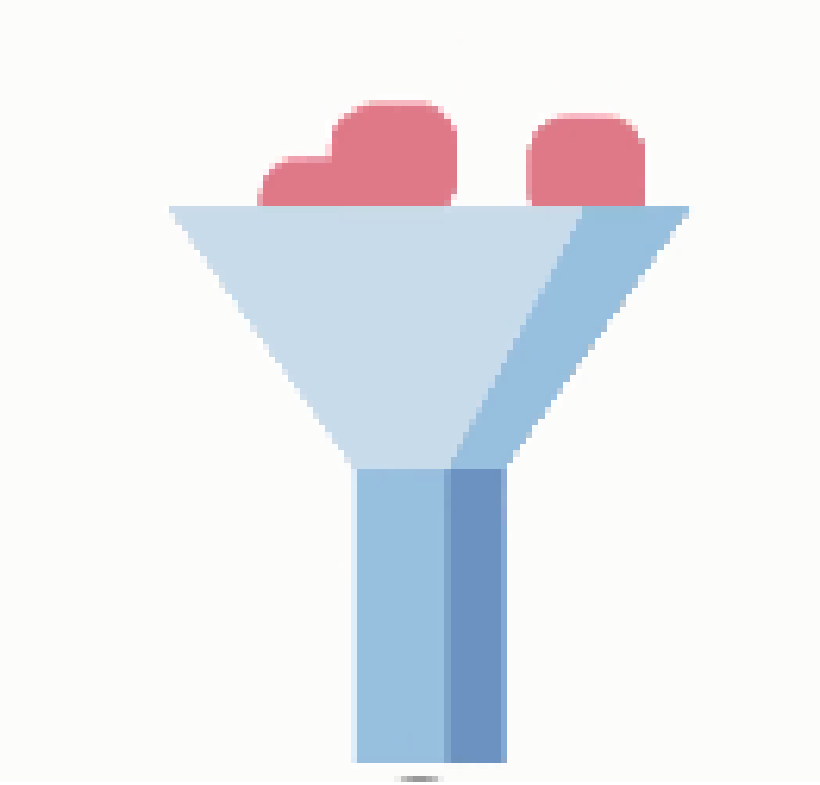


The WHERE clause of the query explains the filtering criteria.

©Simplilearn. All rights reserved.

FilteredRowSet Object

The FilteredRowSet object allows disconnected RowSet objects to filter data without querying the data source.

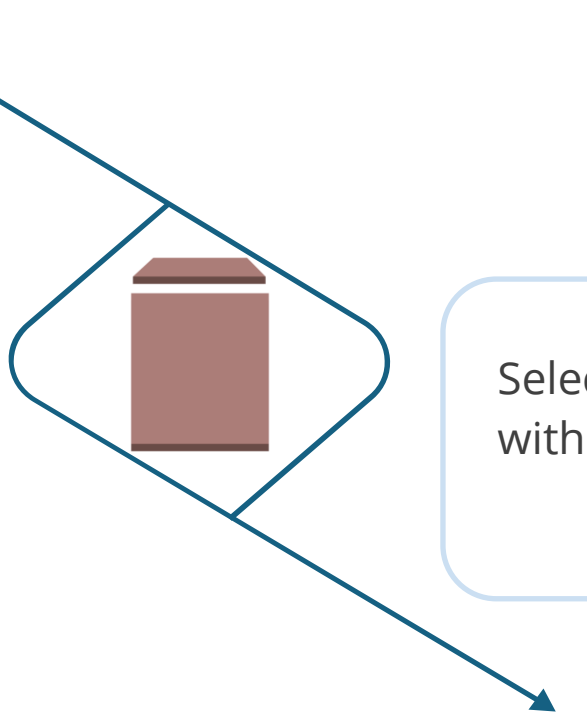


©Simplilearn. All rights reserved.

FilteredRowSet Object

FilteredRowSet object provides the following capabilities:

Visibility of the rows based on the set criteria

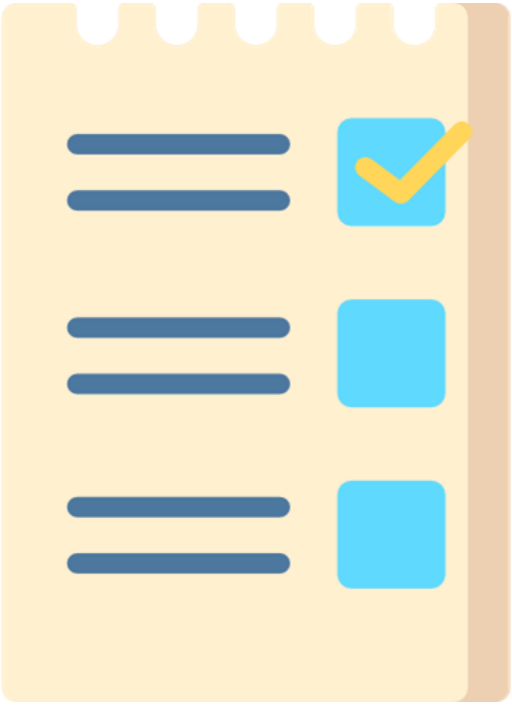


Selection of data to be visible without being connected to a data source

©Simplilearn. All rights reserved.

FilteredRowSet Object

The Predicate interface defines the criteria used to determine which rows will be displayed in the FilteredRowSet object.



©Simplilearn. All rights reserved.

FilteredRowSet Object

The object created with this class is initialized with the following:

- The upper limit of the range in which the values must fall
- The low end of the range in which the values must be contained
- The range of values set by the high and low borders

For the value to be valid, the column name or number must fall within the range specified by the high and low bounds.

©Simplilearn. All rights reserved.

simplilearn

FilteredRowSet Object

Syntax:

```
The following class StateFilter implements Predicate {
Private int city1;
Private int city2;
Private String colname = null;
Private int colnumber = -1;
Public StateFilter ( int city1, int city2, int colnumber );
This.city1 = city1;
This.city2 = city2;
this.colnumber = colnumber;
}
Public StateFilter ( int city1, int city2, String colname ) {
This.city1 = city1;
This.city2 = city2;
this.colname = colname;
}
```

©Simplilearn. All rights reserved.

FilteredRowSet Object

Syntax:

```
Public boolean evaluate ( Object value, String columnname ) {
Boolean evaluation = true;
If ( columnname.equalsIgnoreCase(this.colname)) {
Int columnname = ((Integer) value).intValue();
If ((columnValue >= this.city1)
&&
(columnValue <= this.city2)) {
Evaluation = true;
} else {
Evaluation = false;
}
}
Return evaluation;
}
```

©Simplilearn. All rights reserved.

FilteredRowSet Object

Syntax:

```
Public boolean evaluate( RowSet rs ) {
CachedRowSet frs = (CachedRowSet)rs;
Boolean evaluation = false;
Try {
Int columnValue = -1;
If ( this.colnumber > 0 ) {
columnValue = frs.getInt ( this.colnumber
);
} else if ( this.colname != null ) {
columnValue = frs.getInt( this.colname );
} else {
Return false;
}
If (( columnValue >= this.city1 )) {
evaluation = true;
}
}
```

```
catch ( SQL Exception e )
JDBCUtilities.printSQLException(e);
Return false;
} catch ( Null PointerException npe ) {
System.err.println("NullPointerException
occur");
Return false;
}
Return evaluation ;
}
}
```

©Simplilearn. All rights reserved.

FilteredRowSet Object

FilteredRowsetSample creates a filter that only permits rows with a STORE ID column value between 10000 and 10999, indicating locations in New York.

```
StateFilter myStateFilter = new
StateFilter(10000, 10999, 1);
```

Note

The StateFilter class created applies only to one column.

©Simplilearn. All rights reserved.

FilteredRowSet Object

Parameters are feasible to apply to two or more columns in arrays rather than single values.

```
Public Filter2( Object [] city1, Object [] city2, Object [],
int col_number )  {
This.city1 = city1;
This.city2 = city2;
this.col_number = col_number;
}
```

As a result, the three arrays should have the same number of elements.

©Simplilearn. All rights reserved.

FilteredRowSet Object

Implementing the method evaluation for the Filtered2 object, where the parameters are arrays:

```
Public boolean evaluate (RowSet rowset) {
CachedRowSet crset = (CachedRowSet)rowset;
Boolean bool1;
Boolean bool2;
for ( int i =0;  i< colnumber.length; i++
) {
If ( ( rowset.getObject(colnumber [i] >=
city1[i])&&
( rowset.getObject(colnumber [i] <=
city2[i])  {
Bool1 = true;
} else {
Bool2 = true;
}
If (bool2)  {
Return false;
} else {
Return true;
}
}
}
```

Using Filter2 object implementation, the user can check one column or several columns without developing another implementation.

Pass the Object type, which involves converting the primitive type to its Object type.

©Simplilearn. All rights reserved.

simplilearn

Create FilteredRowSet Object

To build the FilteredRowSet object, use an instance of the RowSetShop, which is created from the class RowSetProvider.

FilteredRowSetSample takes this form:

```
RowSetShop shop = RowSetProvider.newShop();
Try (FilteredRowSet Srs =
shop.createFilteredRowSet())
```

©Simplilearn. All rights reserved.

Create FilteredRowSet Object

The FilteredRowSet object must populate itself with data from the tabular data source, which in the reference implementation is the relational database.

- The FilteredRowSet sample's code snippet specifies the properties required to connect to the database and perform its operation.
- The connection is established using the DriverManager class, which is done for user convenience.
- It is normally preferable to utilize a DataSource that implements JNDI.

©Simplilearn. All rights reserved.

Create FilteredRowSet Object

An example from the FilteredRowSetSample:

```
RowSetShop shop = RowSetProvider.newShop();
Try (FilteredRowSet Srs = shop.createFilteredRowSet()
) {
Srs.setCommand("SELECT * FROM FRUIT_HOUSES");
Srs.setUsername(settings.userName);
Srs.setPassword(settings.password);
Srs.setUrl(settings.urlString);
Srs.execute();
```

©Simplilearn. All rights reserved.

Create FilteredRowSet Object

The following code populates the Srs object with the data stored in the FRUIT_HOUSES table:

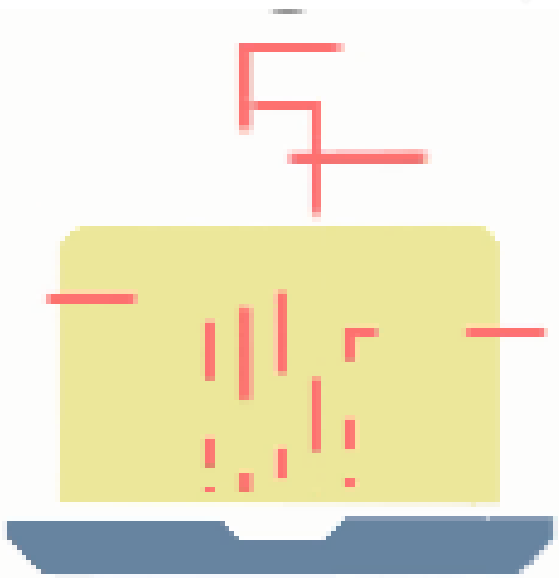
```
Srs.execute();
```

©Simplilearn. All rights reserved.

Create FilteredRowSet Object

By calling the RowSetReader object for Srs, the method:

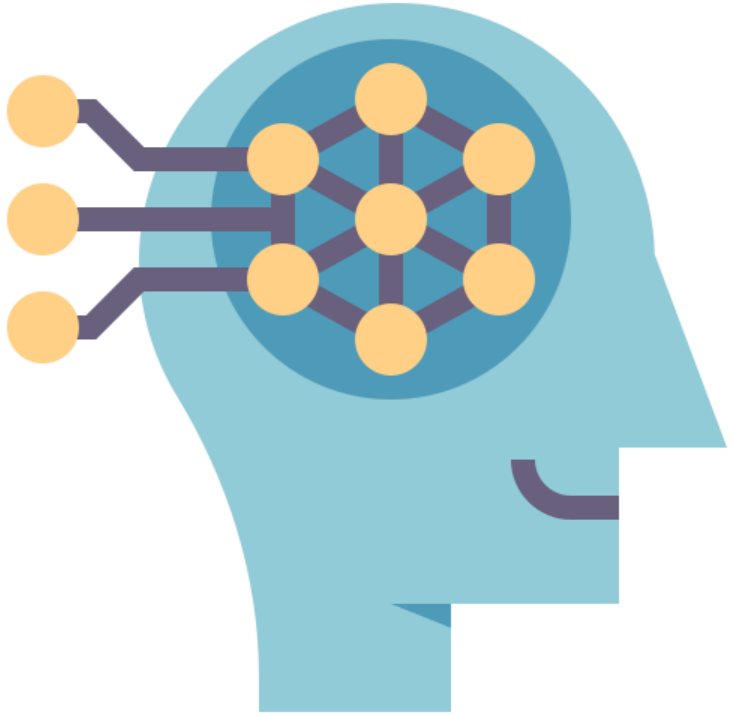
- Creates the connection
- Executes the command for Srs
- Populates Srs with the data from the ResultSet object



©Simplilearn. All rights reserved.

Create FilteredRowSet Object

The CachedRowSet paging methods would have been utilized if the table FRUIT HOUSES had more rows than the Srs object.

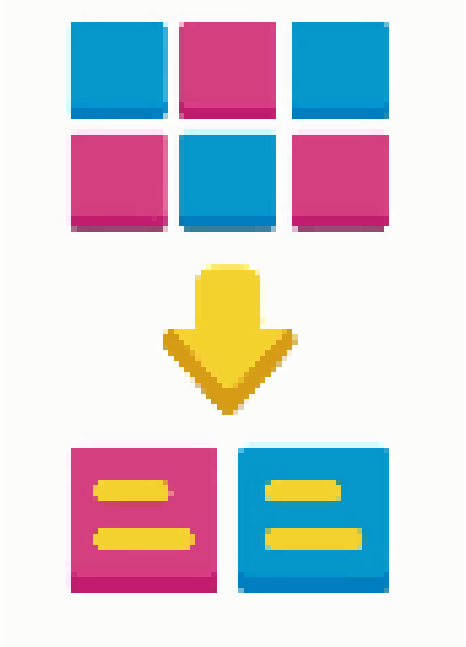


©Simplilearn. All rights reserved.

Creating and Setting Predicate Objects

The Fruit Shop establishments are listed in the FilteredRowSet object Srs.

Users can use selection criteria to reduce the number of rows in the Srs object that are displayed.



©Simplilearn. All rights reserved.

Creating and Setting Predicate Objects

StateFilter class is used to check open stores.

```
StateFilter myStateFilter = new StateFilter(
10000, 10999, 1 );
```

To set the myStateFilter as the filter for Srs, use this syntax:

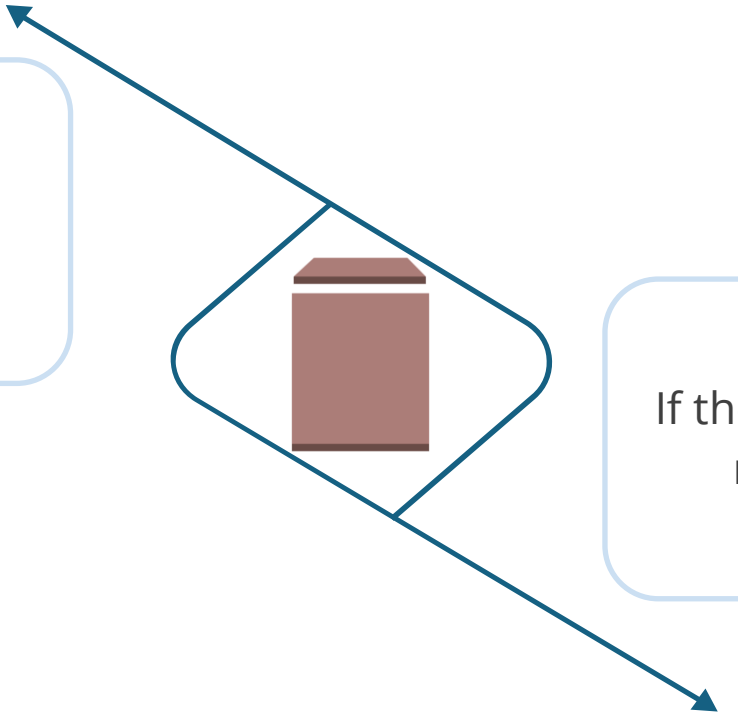
```
Srs.setFilter(myStateFilter);
```

©Simplilearn. All rights reserved.

Creating and Setting Predicate Objects

The reference implementation calls the proper version of the Predicate and evaluates the method described by the user before.

The row will be visible if the return value is true.



If the return value is false, the row will not be visible.

©Simplilearn. All rights reserved.

SettingFilteredRowSet

Multiple filters can be configured in sequential order.

Apply the filtering criteria using setFilter and pass it to the Predicate object.

After invoking the next method on each row, the JDBCRowSet object displays only those that meet the filter criteria.

The setFilter function can be called again by supplying it with a different Predicate object.

©Simplilearn. All rights reserved.

SettingFilteredRowSet

If the owner has a list of Fruit Shop locations in New York by specifying the stateFilter as the Srs.Predicate object:

The business owner must compare the stores in the two cities of New York.

They must write the Predicate implementation that searches for retailers in different cities.

©Simplilearn. All rights reserved.

SettingFilteredRowSet

Syntax:

```
Public class CitiesFilter implements Predicate {
Private String [] cities;
Private String colname = null;
Private int colnumber = -1;
Public CitiesFilter (String [] citiesarg, String colname
arg) {
this.cities = cities arg;
this.colnumber = -1;
this.colname = colname arg;
}
Public CitiesFilter(String [] citiesarg, int
colnumberarg) {
This.cities = cities arg;
this.colnumber = colnumberarg;
this.colname = null;
}
```

©Simplilearn. All rights reserved.

simplilearn

SettingFilteredRowSet

Syntax:

```
Public boolean evaluate Object valuearg, String colname
arg) {
If (colnamearg.equalsIgnoreCase(this.colname)){
for (int cityIdx = 0; cityIdx < this.cities.length;
cityIdx++) {
If
(this.cities[cityIdx].equalsIgnoreCase((String)valuearg))
{
Return true;
}
}
}
Return false;
}
```

©Simplilearn. All rights reserved.

simplilearn

SettingFilteredRowSet

Syntax:

```
Public boolean evaluate(Object valuearg, int
colnumberarg) {
If (colnumber arg == this.colnumber) {
For (int cityIdx = 0; cityIdx < this.cities.length;
cityIdx++) {
If
(this.cities[cityIdx].equalsIgnoreCase((String)valuearg))
{
Return true;
}
}
}
Return false;
}
```

©Simplilearn. All rights reserved.

simplilearn

SettingFilteredRowSet

Syntax:

```
Public boolean evaluate(RowSet rowset) {
If (rowset == null) return false;
Try {
For ( int cityIdx =0; cityIdx < this.cities.length;
cityIdx++ ) {
String cityname = null;
If (this.colnumber > 0) {
cityname = (String)rs.getObject(this.colnumber);
} else if (this.colname != null) {
cityname = (String)rs.getObject(this.colname);
} else {
Return false;
}
```

©Simplilearn. All rights reserved.

simplilearn

SettingFilteredRowSet

Syntax:

```
If (cityname.equalsIgnoreCase(cities[i])) {
Return true;
}
}
} catch (SQL Exception error) {
return false;
}
return false;
}
}
```

©Simplilearn. All rights reserved.

simplilearn

SettingFilteredRowSet

Using this syntax, the FilteredRowSetSample program fragment creates a new filter and iterates through the rows in the Srs, printing the rows with the CITY column.

```
Public void testFilteredRowSet() throw
SQLException {
StateFilter myStateFilter = new
StateFilter(1000, 10999, 1);
String[] cityArray = { "City1", "City2" };
CityFilter myCityFilter = new CityFilter(
cityArray, 2);
RowSetFactory shop =
RowSetProvider.newShop();
Try (FilteredRowSet Srs =
shop.createFilteredRowSet()) {
Srs.setCommand( "SELECT * FROM
FRUIT_HOUSES");
Srs.setUsername(settings.userName);
Srs.setPassword(settings.password);
Srs.setUrl(settings.urlString);
Srs.execute();
System.out.println("\n Before filter:");
FilteredRowSetSample.viewTable(this.con);
```

```
System.out.println("\n Setting state
filter:");
Srs.beforeFirst();
Srs.setFilter(myStateFilter);
this.viewFilteredRowSet(Srss);
System.out.println("\n Setting city
filter:");
Srs.beforeFirst();
Srs.setFilter(myCityFilter);
this.viewFilteredRowSet(Srs);
} catch(SQL Exception e) {
JDBCUtilities.printSQLException(e);
}
}
```

©Simplilearn. All rights reserved.

Extracting Database Information Using Metadata

Problem Statement:

You have been asked to extract information about the database in a Java program using metadata.

Outcome:

You've been tasked with retrieving database information within a Java program using metadata. This involves accessing details like table names, column names, and data types, providing valuable insights for dynamic data handling and application logic.

Note: Refer to the demo document for detailed steps: 08_Extracting_Database_Information _Using_Metadata

ASSISTED PRACTICE

©Simplilearn. All rights reserved.

Assisted Practice: Guidelines

Steps to be followed are:

1. Extract details about the database and list of tables
2. Extract Metadata containing the list of tables

©Simplilearn. All rights reserved.

Key Takeaways

- JDBC is a platform-independent database-to-Java interface.
- JDBC library also provides a complete set of libraries and interfaces that allow smooth access to an underlying database.
- JDBC architecture supports both the two-tier and three-tier processing models for database access.
- To communicate with a database, JDBC drivers implement defined interfaces in the JDBC API.
- CRUD is an acronym that originated from the world of computer programming languages.

©Simplilearn. All rights reserved.

TECHNOLOGY

Thank You

©Simplilearn. All rights reserved.

simplilearn