51)Create a Java program to connect to a MySQL database and retrieve all student records from a Student table using JDBC. Display each student's ID, name, age, course, and marks.

```java
import java.sql.*;

public class Question51 {

    public static void main(String[] args) {

        String url = "jdbc:mysql://localhost:3306/college";

        String user = "root";

        String password = "your_mysql_password"; // Replace with your actual password

        try {

            // Load the MySQL JDBC driver (optional for newer versions)

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

            // Establish connection

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

            // Create statement and query

            Statement stmt = con.createStatement();

            ResultSet rs = stmt.executeQuery("SELECT * FROM Student");

            // Display data

            System.out.println("ID | Name | Age | Course | Marks");

            while (rs.next()) {

                int id = rs.getInt("id");

                String name = rs.getString("name");

                int age = rs.getInt("age");

                String course = rs.getString("course");

                int marks = rs.getInt("marks");

                System.out.println(id + " | " + name + " | " + age + " | " + course + " | " + marks);

            }

            // Close connection

            con.close();

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

}
```

Stream API Activity::

1)class Product::

id:Long

name:String

category:String

price:Double

Solution -1 (Product Class):

```
class Product {
    Long id;
    String name;
    String category;
    Double price;

    public Product(Long id, String name, String category, Double price) {
        this.id = id;
        this.name = name;
        this.category = category;
        this.price = price;
    }

    public String toString() {
        return name + " (" + category + ") - ₹" + price;
    }
}
```

2. class Customer:

id:Long

name:String

tier:Integer

Solution-2 (Customer Class)

```
class Customer {
    Long id;
    String name;
    Integer tier;
    public Customer(Long id, String name, Integer tier) {
        this.id = id;
        this.name = name;
        this.tier = tier;
    }
}
```

3.class Order:

id:Long

status:String

orderDate:LocalDate

deliveryDate:LocalDate

products:List<Product>

customer:Customer

Solution-3(Order Class):

```
import java.time.LocalDate;
import java.util.List;
class Order {
    Long id;
    String status;
    LocalDate orderDate;
    LocalDate deliveryDate;
    List<Product> products;
    Customer customer;
    public Order(Long id, String status, LocalDate orderDate, LocalDate deliveryDate, List<Product> products, Customer
customer) {
        this.id = id;
        this.status = status;
        this.orderDate = orderDate;
        this.deliveryDate = deliveryDate;
        this.products = products;
        this.customer = customer;
    }
}
```

Other Questions:

```
public class StreamApiActivity {

    public static void main(String[] args) {

        List<Product> products = List.of(

            new Product(1L, "Java Programming", "Books", 199.0),

            new Product(2L, "Data Structures", "Books", 99.0),

            new Product(3L, "Baby Diapers", "Baby", 350.0),

            new Product(4L, "Toy Car", "Toys", 450.0),

            new Product(5L, "Lego Set", "Toys", 1200.0),

            new Product(6L, "Baby Powder", "Baby", 150.0)

        );

        List<Customer> customers = List.of(

            new Customer(1L, "Ravi", 1),

            new Customer(2L, "Teja", 2),

            new Customer(3L, "Priya", 2)

        );

        List<Order> orders = List.of(

            new Order(101L, "Delivered", LocalDate.of(2021, 2, 10), LocalDate.of(2021, 2, 15), List.of(products.get(0),
products.get(2)), customers.get(1)),

            new Order(102L, "Shipped", LocalDate.of(2021, 3, 5), LocalDate.of(2021, 3, 10), List.of(products.get(4)),
customers.get(2)),

            new Order(103L, "Delivered", LocalDate.of(2021, 1, 25), LocalDate.of(2021, 1, 30), List.of(products.get(1),
products.get(5)), customers.get(0)),

            new Order(104L, "Delivered", LocalDate.of(2021, 4, 1), LocalDate.of(2021, 4, 5), List.of(products.get(3)),
customers.get(2))

        );


        // Now apply the required queries:
```

1) **Products in category “Books” with price > 100 :**

```
System.out.println("1. Books with price > 100:");

products.stream()

    .filter(p -> p.category.equals("Books") && p.price > 100)
    .forEach(System.out::println);
```

2) **Orders with products in category “Baby”:**

```
System.out.println("\n2. Orders with Baby products:");

orders.stream()

    .filter(o -> o.products.stream().anyMatch(p -> p.category.equals("Baby")))

    .forEach(o -> System.out.println("Order ID: " + o.id));
```

3) **Products in “Toys” category with 10% discount:**

```
System.out.println("\n3. Toys with 10% discount:");
products.stream()
  .filter(p -> p.category.equals("Toys"))
  .map(p -> new Product(p.id, p.name, p.category, p.price * 0.9))
  .forEach(System.out::println)
```

4) **Products ordered by tier 2 customers between 01-Feb-2021 and 01-Apr-2021 :**

```
System.out.println("\n4. Products ordered by tier 2 customers (Feb-Apr 2021):");

    orders.stream()
      .filter(o -> o.customer.tier == 2)
      .filter(o -> o.orderDate.isAfter(LocalDate.of(2021, 1, 31)) &&
             o.orderDate.isBefore(LocalDate.of(2021, 4, 2)))
      .flatMap(o -> o.products.stream())
      .distinct()
      .forEach(System.out::println)
```

5) **Cheapest product in category “Books” (use findFirst) :**

```
System.out.println("\n5. Cheapest Book:");
    products.stream()
      .filter(p -> p.category.equals("Books"))
      .sorted(Comparator.comparingDouble(p -> p.price))
      .findFirst()
      .ifPresent(System.out::println)
```

6) **3 most recent orders (by orderDate descending) :**

```
System.out.println("\n6. 3 Most Recent Orders:");
orders.stream()
  .sorted(Comparator.comparing(Order::orderDate).reversed())
  .limit(3)
  .forEach(o -> System.out.println("Order ID: " + o.id + ", Date: " + o.orderDate));
```

7) **Total amount of orders placed in Feb 2021 :**

```
System.out.println("\n7. Total order amount in Feb 2021:");
double total = orders.stream()
  .filter(o -> o.orderDate.getMonthValue() == 2 && o.orderDate.getYear() == 2021)
  .flatMap(o -> o.products.stream())
  .mapToDouble(p -> p.price)
  .sum();
System.out.println("₹" + total)
```

8) **Summary statistics for “Books” category:**

```
System.out.println("\n8. Statistics for Books:");
    DoubleSummaryStatistics stats = products.stream()
      .filter(p -> p.category.equals("Books"))
      .mapToDouble(p -> p.price)
      .summaryStatistics();

    System.out.println("Count: " + stats.getCount());
    System.out.println("Sum: ₹" + stats.getSum());
    System.out.println("Average: ₹" + stats.getAverage());
    System.out.println("Max: ₹" + stats.getMax());
    System.out.println("Min: ₹" + stats.getMin());
```

9) **Most expensive product by category (use Collectors.maxBy) :**

```
System.out.println("\n9. Most expensive product by category:");
Map<String, Optional<Product>> maxByCategory = products.stream()
  .collect(Collectors.groupingBy(
    p -> p.category,
    Collectors.maxBy(Comparator.comparingDouble(p -> p.price))
  ));
```

```
        maxByCategory.forEach((category, prodOpt) ->
            System.out.println(category + ": " + prodOpt.orElse(null))
        );
    }
}
```