Operation and Automation Solutions

Graduate Trainee Program 2024

Written Test

Application No.: **1651**

Points to Note

Please let us know the link to download zip / git clone your work.

Try Your Best & Enjoy the Process.

Good Luck!

Shoalter
Technology Ltd.
Member of HKTV Group

Q1. (10%)

Write a Java program to reverse a given string without using any built-in methods or libraries.

e.g.

Input: retlaohS

Output: Shoalter

Answer:

```java
public class ReverseString {

    public static String reverse(String text) {
        char[] reversedChars = new char[text.length()];
        int j = 0;
        for (int i = text.length() - 1; i >= 0; i--) {
            reversedChars[j++] = text.charAt(i);
        }
        return new String(reversedChars);
    }

    public static void main(String[] args) {
        String originalText = "retlaohS";
        String reversedText = reverse(originalText);

        System.out.println("Original text: " + originalText);
        System.out.println("Reversed text: " + reversedText);
    }
}
```

Q2. (15%)

Warehouse supervisor requests an hourly report to show how well their packers perform. Based on the sample table and data given below, please write a SQL statement to generate the number of totes being PACKED per user during 10:00:00 to 10:59:59, and provide the expected output.

table name: order_tote_process_log

table definition:

Field	Type	Null	Key	Default	Extra
id	bigint(20)	NO	PRI	NULL	
tote_number	varchar(20)	NO		NULL	
action_code	varchar(255)	NO		NULL	
process_date	datetime	NO		NULL	
user_code	varchar(255)	NO		NULL	

sample data:

id	tote_number	action_code	process_date	user_code
1	123789	START	2023-11-01 09:16:50	SYS
2	199888	START	2023-11-01 09:16:55	SYS
3	124554	START	2023-11-01 09:17:03	SYS
4	123789	PICKED	2023-11-01 09:17:03	SYS
5	123789	PACKED	2023-11-01 10:35:50	P1
6	124554	PICKED	2023-11-01 11:00:50	SYS
7	124554	PACKED	2023-11-01 11:03:50	P2

Answer:

```
SELECT user_code, COUNT(*) AS totes_packed
FROM order_tote_process_log
WHERE action_code = 'PACKED'
  AND process_date BETWEEN '2023-11-01 10:00:00' AND '2023-11-01 10:59:59'
GROUP BY user_code
ORDER BY totes_packed DESC;
```

Expected Output:

user_code	totes_packed
P1	1

Shoalter Technology Ltd.
Member of HKTV Group

Q3. (20%)

Warehouse team wants to know the maximum quantity of a product that can be shipped in a carton box.

Write an executable method/function that outputs the required information.

Sample data:

Name	**Length (mm)**	**Width (mm)**	**Height (mm)**
Carton box	320	260	200
Product (Toblerone 100g)	210	35	35

Answer:

[4]:
```python
def max_quantity_in_carton(carton_dimensions, product_dimensions):

    # Calculate the volumes of the carton and product
    carton_volume = carton_dimensions[0] * carton_dimensions[1] * carton_dimensions[2]
    product_volume = product_dimensions[0] * product_dimensions[1] * product_dimensions[2]

    # Calculate the maximum quantity and round down to the nearest integer
    max_quantity = carton_volume // product_volume
    max_quantity_rounded_down = int(max_quantity)  # Ensure a whole number of products

    return max_quantity_rounded_down

# Sample Data
carton_dimensions = [320, 260, 200]
product_dimensions = [210, 35, 35]

# Calculate and print the maximum quantity
max_quantity = max_quantity_in_carton(carton_dimensions, product_dimensions)
print("Maximum quantity of products in the carton:", max_quantity)
```

```
Maximum quantity of products in the carton: 64
```

Q4. (25%)

Logistics team requests a feature to allow customers to calculate the shipping fee according to the dimension (cm) / weight (kg) / temperature condition / quantity of the parcels. Based on the sample rate card given below, please

1) design an API and draft the API specification for the shipping fee calculation request,

2) develop a workable demo web application.

Assumption:

1. Can have multiple items in single request
2. Will not have items with mixed temperature condition in single request

Volumetric Weight formula: (length x width x height) / 5000

Rate Card:

Weight (kg)	**Temperature Condition**	**HKD (per kg)**
≤ 5	Ambient	$10
> 5	Ambient	$15
≤ 5	Chill	$20
> 5	Chill	$30

* The weight is according to the volumetric weight or physical weight, whichever is higher

Answer:

My Personal github link:
https://github.com/jasonngtt/HKTV-GT-Program.git (Public)

Inside the repository, Please kindly open the attached.
- TestB-Q4-calculate-shipping-fee.json
- TestB-Q4-app.py
- TestB-Q4-app_testcase.json

Q5. (30%)

Prepare a microservice with documentation and user guide in the Kubernetes environment for data migration from CSV to MySQL (MariaDB). After data migration, prepare indexing in the database. And provide the below API endpoint.

API 1 Spec (Search against DATE_OCC):

[POST] Path: <To Be Design By Candidate>
Body:

```
{
  "from_date": "2022-01-01 00:00:00",
  "to_date": "2022-06-01 00:00:00"
}
```

Response:

```
[
 {
  "DR_NO": "String",
  "DATE_RPTD": "String",
  "DATE_OCC": "String",
  "TIME_OCC": "String",
  "AREA": "String",
  "AREA_NAME": "String",
  "Rpt_Dist_No": "String",
  "Crm_Cd_Desc": "String",
  "Vict_Age": "String",
  "Vict_Sex": "String",
  "Vict_Descent": "String",
  "Premis_Cd": "String",
  "Premis_Desc": "String",
  "Weapon_Used_Cd": "String",
  "Weapon_Desc": "String",
  "Status": "String",
  "Status_Desc": "String"
 } ]
```

*Every search period will not more than 6 months

Preferred Microservice Framework: Spring Boot (Latest version, 3.2.1).
Java Version: Any version compatible with the application.
DB normal form: 3NF

Please note that the main file is quite large, the method for feeding the data to the microservice is worth consideration. <Cont'd next page>

Data Source (MIT License) https://www.kaggle.com/datasets/manjitbaishya001/la-crime-data-2010-to-2023/data?select= mo_codes.csv

Answer:

I try to answer the question. I am not going to clone the file on github.

We first download the data from Kaggle which provided above, we get

- la_crime_2010_to_2023.csv (782.5 MB)
- mo_codes.csv (24 KB)

Now I have to initialize Spring Boot Project and create a Model Class for representing the data model. I will name it as '**CrimeData.java'**, and also create a '**CrimeDataController.java'** controller class with endpoints for data migration.

The main file size **la_crime_2010_to_2023.csv** is really large, we can handle it with Batch processing. Now here's the data migration logic in **CrimeDataController**

```
@RestController
@RequestMapping("/api")
public class CrimeDataController {

    private final CrimeDataService crimeDataService;

    @Autowired
    public CrimeDataController(CrimeDataService crimeDataService)
{
        this.crimeDataService = crimeDataService;
    }

    @PostMapping("/migrate-data")
    public ResponseEntity<String> migrateData(@RequestBody
List<CrimeData> crimeDataList) {
        // Implement batch processing logic
        int batchSize = 1000; // Choose an appropriate batch size

        for (int i = 0; i < crimeDataList.size(); i += batchSize)
{
            int endIndex = Math.min(i + batchSize,
crimeDataList.size());
            List<CrimeData> batch = crimeDataList.subList(i,
endIndex);

            // Process the current batch and save to the database
asynchronously
            crimeDataService.processAndSaveAsync(batch);
        }

        return ResponseEntity.ok("Data migration started. Check
logs for progress.");
    }
}
```

```
@PostMapping("/migrate-data")
public ResponseEntity<String> migrateData(@RequestBody
List<CrimeData> crimeDataList) {
    // Implement CSV to MySQL data migration logic
    // Use Spring Data JPA or JDBC to save data to MariaDB
    // Return appropriate response
}
```

Now, we setup the MariaDB (MySQL) Database, we dockerize the application and build docker image. New, we create Kubernetes Deployment YAML and named '**deployment.yaml**' to define the Kubernetes deployment and '**service.yaml**' to define a Kubernetes service for the microservice.

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-microservice
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-microservice
  template:
    metadata:
      labels:
        app: my-microservice
    spec:
      containers:
        - name: my-microservice
          image: my-microservice:latest
          ports:
            - containerPort: 8080
```

service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: my-microservice
spec:
  selector:
    app: my-microservice
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
  type: LoadBalancer
```

Now, the created User guide are uploaded on Github.

My Personal github link:
https://github.com/jasonngtt/HKTV-GT-Program.git (Public)

Inside the repository, Please kindly open the attached.

- **TestB_Q5_User Guide in Kubernetes environment.txt**

Next, we will test the microservice by sending data migration requests to the specified endpoint and add indexing to MariaDB (MySQL) Database.

Plus, we need to modify Microservice for Search Endpoint by updating '**CrimeDataController.java'** class to include a new endpoint for searching based on 'DATE_OCC'

```
@RestController
@RequestMapping("/api")
public class CrimeDataController {

    // Existing code...

    @PostMapping("/search-by-date")
    public ResponseEntity<List<CrimeData>>
searchByDate(@RequestBody DateRangeRequest dateRangeRequest) {
        // Implement search logic using the provided date range
        // Query the database and return the results
    }
}

public class DateRangeRequest {
    private LocalDateTime fromDate;
    private LocalDateTime toDate;

    // Getters and setters
}
```

We also implement the Search logic by using '**searchByDate**' method and add a method in '**CrimeDataService.java**' for querying the database by date range

```
@PostMapping("/search-by-date")
public ResponseEntity<List<CrimeData>> searchByDate(@RequestBody
DateRangeRequest dateRangeRequest) {
    LocalDateTime fromDate = dateRangeRequest.getFromDate();
    LocalDateTime toDate = dateRangeRequest.getToDate();

    // Implement database query using Spring Data JPA or JDBC
    List<CrimeData> result =
crimeDataService.findByDateRange(fromDate, toDate);

    return ResponseEntity.ok(result);
```

```
}

public List<CrimeData> findByDateRange(LocalDateTime fromDate,
LocalDateTime toDate) {
    // Implement database query using Spring Data JPA or JDBC
    // Use the provided date range in the query
    // Return the results
}
```

At last, we test the Search Endpoint by using '**curl**' and deploy to Kubernetes.