- Write a SQL query to retrieve the top 5 customers who have spent the most in the last year from an `orders` table. The `orders` table contains the following columns: `order_id`, `customer_id`, `order_date`, and `order_total`. Assume the current date is `2024-08-14`. You should account for customers who may have multiple orders. The query should return the customer ID and the total amount spent in descending order.

Select top 5 customer_id, Sum(order_total) As total_spending from orders where order_date >= dateadd (YEAR, -1, GETDATE()) group by customer_id order by total_spending desc

- Describe how you would prioritize and plan the following support tickets:
    - A customer reports a system outage affecting multiple users.
    - A customer requests a password reset.
    - A customer reports a minor UI bug.
    - A customer requests help with a new feature.

When prioritizing and planning support tickets, I ensure that critical issues are handled promptly while addressing other requests in a timely manner. Here's my approach to handling the listed tickets:

1. System Outage Affecting Multiple Users

- ➔ *Highest priority.*
    - ◆ *Immediate action is required to address the outage, diagnose the issue, and resolve it swiftly. After resolution, review the incident to prevent future occurrences.*

2. Password Reset Request:

- ➔ *Medium priority*
    - ◆ *Verify the requestor's identity, reset the password promptly, and confirm with the user that they have regained access.*

3. Minor UI Bug Report:

- ➔ *Low to medium priority*
    - ◆ *Assess the bug's impact, plan to fix it in a future update, and inform the customer that their report will be addressed.*

4. Help with a New Feature Request:

- *Medium priority*
    - *Understand the user's needs, provide support or instructions, and follow up to ensure the feature is being used effectively.*

- Explain how you would diagnose a situation where a database query that previously ran in under a second is now taking several minutes to complete.

- *Verify the Issue: Confirm the query's performance problem and check if it is consistent.*
- *Check if there's any recent changes: Look into recent changes to the database, application code, or data volume.*
- *Examine Indexes: Ensure indexes are correctly implemented and used.*
- *Monitor Database Performance: Check server resource utilization and review database logs.*
- *Optimize the query: Rewrite the query or use hints to improve performance.*
- *Test and Validate:Compare the optimized query's performance and ensure results are correct.*
- *Review and Document: Record findings and changes, and set up monitoring for future issues.*

- How would you approach handling a recurring error in a production environment that affects multiple users but has not been reproducible in the testing environment?

Here's my approach to address a recurring production error that affects multiple users but isn't reproducible in testing, gather detailed information and error logs, analyze the issue under production-like conditions, enhance monitoring, isolate the problem, collaborate with relevant teams, implement and validate a fix, and document the incident for future prevention.

- Describe how you would use JIRA to track a support ticket from the time it's created to the time it’s resolved. Include statuses you would use, how you ensure proper communication and documentation at each stage.

Here’s my approach on how I am going to track a support ticket in JIRA from creation to resolution, start with the ticket in the Open status, then move it to In Progress for initial assessment, update it to Ready for Testing or Resolved once a fix is implemented, and finally close it with the Closed status, at every stage, maintain effective communication by regularly updating ticket comments with pertinent information, informing stakeholders about progress, and offering clear documentation to aid future reference and improvements.

- What steps would you take to optimize a slow-running SQL query? Provide at least three different approaches you could try.

We can take several approaches to improve performance. Here are three effective strategies in my opinion:

➔ *Analyze query execution: Examine and refine the query’s execution plan to eliminate inefficiencies.*
➔ *Improve Indexing: Assess and implement suitable indexes to enhance query performance.*
➔ *Data optimization: Revise table structures, consider partitioning large tables, and ensure statistics are current.*

- How would you use Postman to test an API endpoint that requires both a specific header and a JSON body? Include the steps you would take to validate the *response.*

➔ *Create a New Request: Open Postman and configure a new request with the appropriate HTTP method and URL.*
➔ *Add Header: Include the necessary header.*
➔ *Add JSON Body: Input the JSON payload in the Body tab, selecting raw and JSON as the format.*
➔ *Send Request: Click Send to submit the request.*
➔ *Validate Response: Review the status code, examine the response body, and check*

the response headers; use the Tests tab for automated validation if desired.

- An user is experiencing intermittent connectivity issues with our application that hosted on AWS. What steps would you take to troubleshoot and resolve the issue?

To be able to troubleshoot intermittent connectivity issues with an AWS-hosted application effectively, the steps I will do are to gather user reports and logs, check AWS service health and network configurations, monitor application and database performance, test network latency, review code and configurations, address user-specific concerns, implement and test fixes, and communicate with users while documenting the resolution process.

Database Investigation:

- **Scenario:** A user reports that when using the company's web application to search for orders placed within the last month, the results are missing some records that should be included. The user is particularly concerned because they know of specific orders that should appear but do not. Your task is to investigate and resolve the issue.
- **Table Schema:**
    a. `orders` Table:
        - `order_id` (INT, Primary Key)
        - `customer_id` (INT, Foreign Key referencing `customers` table)
        - `order_date` (DATETIME)
        - `order_total` (DECIMAL)
        - `status` (VARCHAR, e.g., 'completed', 'pending', 'cancelled')

    b. `customers` Table:
        - `customer_id` (INT, Primary Key)
        - `customer_name` (VARCHAR)
        - `email` (VARCHAR)
        - `country` (VARCHAR)
- **Task:**

a. Write a SQL query or queries to investigate the issue and explain the possible reasons why records might be missing. Document the steps you would take to identify the root cause, and propose a solution to fix it. (You can make assumptions for information not given)

- ➔ *I need to verify the data first. This sample query gets all orders from the past month to confirm whether the missing records are present in the database.*

Select order_id, customer_id, order_date, order_total, status from orders where order_date >= dateadd (MONTH, -1, getdate ()) order by order_date

- ➔ *I need to verify also if there was an order discrepancy. This sample identifies any discrepancies in order statuses that could impact their visibility in search results.*

Select status, count(*) from orders where order_date >= dateadd (MONTH, -1, getdate()) group by status

- ➔ *I also need to check the other related tables. This query identifies orders which could signal potential data integrity problems.*

Select o.order_id, o.customer_id, o.order_date, o.order_total, o.status from orders o left join customers c ON o.customer_id = c.customer_id where o.order_date >= dateadd(month, -1, getdate()) and c.customer_id is NULL;

Conclusion: The possible reason for the missing records might be the date range issues occur if the application uses a different range or format that excludes some orders, filter or search logic could be excluding records due to additional conditions, data integrity issues might arise if orders are missing customer information.

Solution: To resolve the issue, we need to ensure the application query matches the SQL query used for direct database checks, including the correct date range and filters, address any missing or inconsistent data by updating records and verifying customer references, test the application with the revised logic to confirm that all expected records appear in the search results while documenting findings and changes to prevent future problems.

Support Case Study:

- **Scenario:** A critical production bug is reported by a high-priority client. The end user is using a feature in our web application that generates stats dashboard based on specific data inputs. The user reports that when they try to generate the dashboard, the system either times out or returns incomplete data. This feature is crucial for the customer's operations, and they have a major deadline approaching. The initial ticket provides limited information: "Stats dashboard is failing; need urgent fix."

- **Additional Context:**
  - The web application has recently undergone a minor update that involved changes to the analytic module.
  - The customer is accessing the application during peak usage hours, and you have noticed similar but non-critical issues reported by other users in the past week.
  - The dashboard feature interacts with several backend services, including a database and an external API for fetching additional data.
- **Task:** Outline a step-by-step investigation plan, including how you would gather additional details from the customer, diagnose the root cause using tools such as monitoring logs and SQL queries, and manage communication with the customer to keep them informed and manage their expectations. Also, propose both immediate and long-term solutions to mitigate the issue and prevent future occurrences. (You can make assumptions for information not given)

Here are some of the approaches I would consider for further investigation:

➔ *Gather informations:*

a. *Reach out to the Customer: Check the client to collect more information about the issue, including the exact steps taken, specific data inputs used, any error messages or screenshots received, and the frequency of the problem and get any relevant error logs or screenshots from the customer if necessary.*

➔ *Initial checking:*

a. *Check System Status: Assess the web application's status to ensure all components are functioning properly and to determine if the issue is related to peak usage times or high load.*

➔ *Confirm if there are recent system updates:*

a. *Review Changes: Investigate recent updates to the analytic module for possible bugs or incomplete implementations affecting the dashboard. If* *the update is identified as the cause, consider rolling it back to see if it resolves the issue.*

- ➔ *Investigate further using monitoring and logs:*
    a. *Application Logs: Examine application logs for error messages or stack traces related to the dashboard feature. Look for recurring errors that might indicate a specific problem.*
    b. *Database Logs: Run queries to check for performance issues or timeouts in the database and review the execution plan for dashboard-related queries to identify inefficiencies.*
    c. *External API Logs: Verify if the external API is returning incomplete or delayed data and ensure API calls are successful and within expected time limits.*

- ➔ *Workaround Solutions: We need to temporarily optimize SQL queries or increasing database resources to handle peak loads, adjusting timeout settings for database and API connections if necessary, and rolling back recent updates if they are identified as the cause.*

- ➔ *Permanent Solutions: We need to involve developing and implementing long-term solutions such as conducting a thorough code review and refactoring the modules to address the root cause, enhancing the dashboard feature and related backend services for improved performance under high loads, establishing comprehensive monitoring and alerting for database performance, API health, and application stability, and performing load testing to ensure the application can effectively handle peak usage.*

- ➔ *Document: Lastly, we need to document all the findings, steps taken, and solutions implemented for future reference and continuous improvement, and apply lessons learned to prevent similar issues by improving the update testing process and enhancing monitoring tools.*