Elaborate how you tested your internship or academic projects.
a. What did the system used to do?
b. What other systems have you seen in the wild like that?
c. How do you approach the testing problem?
d. What were interesting bugs?
e. How did you fix them?

a. The system i used for the testing are **Control-m workload automation** tool to run the job regarding the loading tables with the data, **Putty** for running unix commands,**Winscp** for the storage location of the data,**Data Analytic Studio** for running the sql queries,**HP ALM** tool to upload test cases , execute the test cases , raise & manage & track the defects, **Jira** for the management for the on going tranche. We used an Infosys developed tool called **IDTW** for automating the test case.

b.

c. The approach for testing starts with **preparing the test cases** according to the requirements given, then **start with the query preparation according** to the new requirements or the changes done in the existing ones. Make sure that we **add both positive and negative test cases** according to the requirements.

d. We **raise the defects according to the severity and the priority** for which they need to be fixed. Most of the defects were the confluence page not updated properly by the dev team which were low - medium priority. Some defects were that the source data and target data were different which were on medium - high priority with high severity. The defects are logged into the ALM tool and the defect cycle is tracked over there.

e. The most of the defects were fixed after the inputs from the dev team as they update the confluence page for the reference.

Banking works by transferring money from account A to account B. Most of the time account A is in one bank while account B is another bank.
Suppose someone writes an implementation for such a money transfer.
1. What are the test cases?
2. What are the issues in such a system?
3. What can we do to mitigate some of the issues ?
4. Write the code yourself to demonstrate the mitigations.

1. Creating test cases for transferring an amount from one bank to another bank involves testing various scenarios to ensure the process works correctly. Here are some test cases you could consider:

a.Test transferring a valid amount from one bank account to another bank account successfully without any issues.

b.Test transferring an amount that exceeds the available balance in the sender's account, Then that should result in a failure with an appropriate error message.

c.Test transferring a negative amount, which should be rejected with an error message indicating an invalid transaction.

d.Attempt to transfer zero amount, which should be rejected with an error message as it is not a valid transaction.

e.Test transferring to an invalid or non-existent account number, Then transaction should fail with an appropriate error message.

f.Test entering incorrect bank details for the recipient account, Then transfer should be rejected with an appropriate error message.

g.Attempt to execute the same transfer transaction twice. The system should prevent the second transaction and display an appropriate error message.

h.Test transferring an amount that exceeds the transaction limit set by the bank, Then transaction should be rejected with an appropriate error message.

i. Check if the correct transaction fee (if any) is applied during the transfer process.

j. Ensure that the sender receives a confirmation message/notification after a successful transfer.

k.After the transfer is completed, check if the transaction details are accurately recorded in the transaction history for both sender and recipient.

2. Transferring money from one bank to another involves a complex process that relies on multiple systems, networks, and security measures. Several issues can arise during the transfer process, potentially causing delays, errors, or failures. Here are some common issues that can be faced while transferring an amount from one bank to another:

Insufficient Funds: If the sender's account does not have enough balance to cover the transfer amount and any associated fees, the transfer will be declined.

Invalid Account Information: Providing incorrect or outdated account details for the recipient can lead to the transfer being sent to the wrong account or rejected by the receiving bank.

Technical Glitches: Technical issues with the banking systems, payment gateways, or networks can cause delays or failures in processing the transfer.

Network Connectivity Problems: Poor network connectivity or outages can disrupt communication between banks and lead to transaction failures.

Transaction Limits: Many banks impose transaction limits on the amount that can be transferred in a single transaction or within a specified timeframe. Exceeding these limits can result in the transfer being rejected.

Security Measures: While essential for safeguarding transactions, security measures such as two-factor authentication or fraud detection systems might occasionally cause legitimate transfers to be flagged or delayed.

Banking Hours and Holidays: Transfers initiated outside of banking hours or on bank holidays may be queued for processing on the next business day, leading to delays.

Payment Processing Time: Interbank transfers often involve intermediary banks, and the processing time can vary depending on the number of intermediaries involved.

Transaction Fee Deduction: If there are transaction fees associated with the transfer, the sender's account may be debited for the fee amount, which could impact the final transferred amount.

Bank-specific Policies: Each bank may have its own policies and restrictions for fund transfers, which could affect the transfer process.

3. To mitigate issues while transferring money from one bank to another, various measures can be taken by banks, financial institutions, and customers. Here are some strategies to help mitigate potential problems:

Double-Check Account Information: Always verify the recipient's account details, including the account number and bank's SWIFT code or routing number, before initiating the transfer. Automated validation tools can help prevent errors in account information.

Maintain Sufficient Funds: Ensure that the sender's account has enough balance to cover the transfer amount and any associated fees to avoid insufficient funds issues.

Transaction Limits Awareness: Inform customers about transaction limits, both per transaction and per day, to avoid exceeding these limits accidentally.

Provide Clear Communication: Banks should communicate clearly with customers about the transfer process, including expected processing times, any potential delays, and relevant fees.

Implement Transaction Verification: Utilize multi-factor authentication and verification processes to ensure the authenticity of the transaction and minimize fraud risks.

Automated Monitoring for Fraud Detection: Implement automated systems to monitor transactions for suspicious activities or patterns, which can help in detecting and preventing fraudulent transfers.

Offer Real-Time Notifications: Provide customers with real-time notifications and updates on their transaction status to keep them informed and aware of any potential issues.

Regular System Maintenance: Conduct regular system maintenance to ensure the banking infrastructure remains stable, secure, and up-to-date, reducing the chances of technical glitches.

Transparent Fee Structure: Clearly outline the fees associated with different types of transfers, including currency conversion fees, to avoid surprises for customers.

4.

```
def verify_recipient_details(recipient_account, recipient_bank):

    if recipient_account and recipient_bank:
        return True
    return False

def check_sufficient_balance(sender_account, transfer_amount):

    sender_balance = 10000
    if sender_balance >= transfer_amount:
```

```
        return True
    return False

def check_transaction_limits(transfer_amount, daily_limit):

    if transfer_amount <= daily_limit:
        return True
    return False

def perform_transaction_verification():
    print("Performing transaction verification...")
    return True


def transfer_amount(sender_account, recipient_account, recipient_bank, transfer_amount):

    if not verify_recipient_details(recipient_account, recipient_bank):
        print("Error: Invalid recipient account or bank details.")
        return

    if not check_sufficient_balance(sender_account, transfer_amount):
        print("Error: Insufficient balance in the sender's account.")
        return

    daily_transaction_limit = 10000
    if not check_transaction_limits(transfer_amount, daily_transaction_limit):
        print("Error: Transfer amount exceeds the daily transaction limit.")
        return

    if not perform_transaction_verification():
        print("Error: Transaction verification failed.")
        return

    transaction_data = {
        "sender_account": sender_account,
        "recipient_account": recipient_account,
        "recipient_bank": recipient_bank,
        "amount": transfer_amount,
    }

    print("Transfer successful. Amount transferred:", transfer_amount)


transfer_amount("Sender_Account", "Recipient_Account", "Recipient_BankXYZ",5000)
```

Write a program to add two integers to be taken as string and which returns a string and come up with the test cases to verify the output
function string addNumbers (string val_a, string val_b)

```
def add(num1_str, num2_str):
    try:
        num1 = int(num1_str)
        num2 = int(num2_str)

        result = num1 + num2

        result_str = str(result)

        return result_str

    except ValueError:
        return "Error: Invalid input. Please provide valid integers as strings."


# Input from the user
num1_str = input("Enter the first integer as a string: ")
num2_str = input("Enter the second integer as a string: ")

sum_str = add(num1_str, num2_str)

print("Sum:", sum_str)
```

Test cases covering various scenarios:

a. Tests a valid addition of two positive integers as strings.
b. Tests addition with one of the integers as negative.
c. Tests addition with one of the integers as zero.
d. Tests with invalid input (non-numeric strings).
e. Tests with large integers.