1.Explain your understanding in detail on what the below query is trying to do and the concepts involved

So above mentioned the question related to query

The query is working with currency exchange rates stored in a table named CURR.

- The aim is Retrieve the most recent and historical exchange rates for specific currency pairs(i.e FromCurrency to ToCurrency), grouped by CurrencyType.
- Then calculating the expirationdate for each exchange rate, based on the next available rate.

  Here Used CTE (Common table expression) is a feature in SQL that allows to create a temporary, named result set that can be referenced within the main SQL query.
  A CTE is helps break down complex queries into smaller readable parts.

  CurrencyType: ti specify the type/category of exchange rate.
  FromCurrency: Source currency
  ToCurrency: Target currency
  Date: The date of the exchange rate.
  CurrencyRate: The exchange rate value.

  The ROW_NUMBER() adds each row, partitioned by CurrencyType , FromCurrency and ToCurrency within each partition, row are ordered by Dte in desending order. The most recent date gets ROW_NUMBER() = 1.

```
WITH cte AS (
        SELECT
              c.currencytype,
              c.fromcurrency,
              c.tocurrency,
              c.date,
              c.currencyrate,
              ROW_NUMBER() OVER (
              PARTITION BY c.currencytype, c.fromcurrency,
      c.tocurrency ORDER BY c.date DESC
      ) AS rownum
       FROM CURR c
       )
```

ExchangeRateType: Same as CurrencyType.

FromCurrency and ToCurrency: Source and target currencies.

ExchangeRate: The currency convension rate.

EffectiveDate : The date this rate becomes effective.

ExpirationDate: The date rate expires.If no next rate exists.

The query performs a LEFT JOIN on the cte itself.

```
SELECT
            t.currencytype AS ExchangeRateType,
            t.fromcurrency AS FromCurrency,
            t.tocurrency AS ToCurrency,
            t.currencyrate AS ExchangeRate,
            t.date AS EffectiveDate,
            CASE
                        WHEN cte.date IS NULL THEN '29990101'
                         ELSE cte.date
                         END AS ExpirationDate
                    FROM cte t
                    LEFT JOIN cte cte
                            ON cte.rownum = t.rownum + 1
                            AND cte.currencytype = t.currencytype
                            AND cte.fromcurrency = t.fromcurrency
                            AND cte.tocurrency = t.tocurrency
                     WHERE t.currencytype = 'MAR';
```

Cte.rownum = t.rownum + 1 : It Ensures pairing with the next row in the same partition

If the next rate(cte.date) does not exist (i.e NULL), it assigns a default expiration date of 2999-01-01

Otherwise it uses the date of the next rate.

By filtering on CurrencyTpe = “MAR” the query limits result to exchange rates of type “MAR”.

2.Write an SQL query to find the average salary from a employee table without including the highest and lowest salaries.

```
SELECT AVG(Salary) AS AverageSalary
FROM Employee
WHERE Salary NOT IN (
   (SELECT MAX(Salary) FROM Employee),
   (SELECT MIN(Salary) FROM Employee));
```

3.Provide a Python script that attempts to open a file named data.csv and read its contents. Include error handling to manage cases where the file does not exist or is unreadable. Explain your error handling strategy.

```
def read_csv_file(data.csv):
   try:
      with open(data.csv, 'r') as file:
           contents = file.readlines()
            print("File contents:")
            print("File contents:")
            for line in contents:
                  print(line.strip())
    except FileNotFoundError:
        print(f"Error: The file '{file_name}' does not exist.")
    except PermissionError:
         print(f"Error: Permission denied when trying to access '{file_name}'.")
    except Exception as e:
         print(f"An unexpected error occurred: {e}")
 file_name = 'data.csv'
read_csv_file(data.csv)
```

Error Hndling Strategy:

FileNotFoundError: This exception occurs when the file specified(data.csv) does not exist in the directory

PermissionError: This exception occurs if the file exist but the user does not have the necessary permission to read it.

4.What are environmental variables in python? How do you setup a virtual environment in python? and why do we have to create a virtual environment in python?

Environment variable are key-value pairs stored in the operating system that influence how process run.In Python these variable can be accessed using os module.

A virtual environment is an isolated python environment that allows you to install mange packaged without affecring the global python installation.

Thease are the steps to create a virtual Environment:

1.Python3 -m venv –help

Most python version come with venu pre installed

2.python3 -m venv myenv

Here this is the creating the environment

Myenv Is the environment variable name

3. source myenv/bin/activate

Active the virtual environment

We are creating virtual environment in python is

Avoid Package conflicts: Different projects might require different version of the same package.A virtual environment keeps dependencies isolated for each project.

We can saftly insatll, test, or update libraries in a virtual environment without affecting the system-wide python or other projects.

Environment variable: External configuration setting accessed through the os module.

5.If you are tasked to parse the below JSON response and load it to a database, How would you structure the data into the database? - would you be able to load it to a single table or should it be loaded as multiple tables? Provide the reason for your choice. - Provide the table name/names with columns that could possibly be primary Key and foreign keys, provide the reason for your choice.

The JSON can be divided into

Orders, Customers, Items

Orders: Contains order-level information(order_id, order_date, total_amount).

Customers: Contains details about the customer(customer_is, name, email).

Items: A list of items the order, each havinf details like item_id, product_name, quantity and price.

Determine the Datanase Design

The JSON is hierarchical form so storing the data in multiple related tables

Date Normalization : avoiding the redundancy by separating customer and item details.

Sccalability: its easier to add or modify entities without affecting others

1.**Orders Table:**

Primary key: order_id

2.Customers Table:

Primary key: customer_id

Foreign key in orders table: Links an order to a customer.

3.Order_Items Table:

Primary key :Composite key (order_id, item-id).

Foreign key: order_id

1.Parsse the JSON Data

The json file is read into memory using a library like json in python.

The data is passed into a python dictionary or list of dictionaries, which allows you to access individual fields programmatically

So this step converts the hierarchical JSON structure into a formate that can be iterated over and processed.

2. Establish a Database Connection

A connection is needed to interact with the database for operations like inserting, updating, or querying data.

3.Map JSON Key to Databse Columns

Each key in the JSON object is mapped to a corresponding column in the database table.

This mapping ensures that the JSON structure aligns with the relational schema, preserving relationships and hierarchy.

4.Insert data into parent tables first

Parent records must exist before child records that reference them via foreign keys.

5.Insert Data into Child Tables

This maintains referential integrity, ensuring that child records are properly associated with their parent records.

6.Handle Duplicates Gracefully

Prevents redundant data, which can lead to storage inefficiencies and data inconsistencies.

7.Automate the process

This approach scales well for large or complex JSON data and reduces human error.

8.Verify Data Integrity

Ensures the data in the database matches the original JSON structure and no data was lost or misaligned during the loading process.