Explain

Well formed XML – It is syntactically correct, starts with an XML declaration,

contains only one root element and has matching opening and closing tags.

Infoset – an abstract representation of XML documents. Infoset contains only

significant information. Cannot represent XML fragments and doesn’t care about the

validity of XML document or associated DTD or XML schema.

JSON schema – used to specify rules for JSON objects. JSON schema uses JSON objects, JSON schema is a JSON.

Shredding – breaking up XML into its nodes and data. Commonly done in order to put data into a relational DB.

Processing instruction – special type of XML node with format <?name content>.

Used to provide information to XML processor(how to handle XML).

XML declaration is processing instruction.

Scalable vector graphics – XML based language for representing vector graphics, SVG defines elements and attributes for describing circles ,rectangles, ellipses, lines and text.

And properties for their coordinates, dimensions, colors, attributes etc.

Question 6 explanation

**Question Breakdown**

The problem involves a relational database model consisting of two tables: Bank and AccountType.

**Tables and Their Columns:**

1. **Bank**
   * code: The unique identifier for a bank (STRING, NOT NULL).
   * name: The name of the bank (STRING, NOT NULL, with an alternate key constraint AK1).
2. **AccountType**
   * bank: Foreign key referring to the code in the Bank table (STRING, NOT NULL).
   * name: Name of the account type (STRING, NOT NULL).
   * freeWithdrawals: Indicates if free withdrawals are allowed (BOOLEAN, NOT NULL).
   * rates: An XML column containing interest rate data (XML, NOT NULL).

**XML Structure:**

The rates column in the AccountType table stores XML data that follows a specific Document Type Definition (DTD):

<!ELEMENT Data (Period+)>

<!ELEMENT Period (InterestRate+)>

<!ATTLIST Period StartDate CDATA #REQUIRED EndDate CDATA #IMPLIED>

<!ELEMENT InterestRate EMPTY>

<!ATTLIST InterestRate MinimumBalance CDATA #REQUIRED Rate CDATA #REQUIRED>

<Data>

<Period Startdate >

<InterestRate MinimumBalance Rate/>

</Period>

* Data contains one or more Period elements.
* Each Period has attributes StartDate and EndDate.
* Each Period contains one or more InterestRate elements.
* Each InterestRate has attributes MinimumBalance and Rate.

**Constraints and Rules:**

* There is always at most one Period without an EndDate.
* No two periods may overlap.
* No two InterestRate elements inside the same Period may have the same MinimumBalance.
* StartDate and EndDate are valid dates.
* MinimumBalance and Rate are valid numbers.

**Task:**

Write an SQL query to:

1. Retrieve information about the highest interest rate per bank.
2. Ensure the result includes one row per bank with the following columns: Code, Name, and HighestRateEver.

**Detailed Solution Steps:**

1. **Extract XML Data:**
   * Parse the XML stored in the rates column to access the Rate attributes within InterestRate elements.
   * Use SQL XML functions to achieve this.
2. **Find the Highest Interest Rate:**
   * Once the rates are extracted, identify the maximum rate for each bank.

Here is the detailed SQL query to solve the problem:

WITH RateData AS (

SELECT

Bank.code,

Bank.name,

CAST(x.value('(InterestRate/@Rate)[1]', 'FLOAT') AS FLOAT) AS Rate

FROM

Bank

JOIN AccountType ON Bank.code = AccountType.bank

CROSS APPLY AccountType.rates.nodes('/Data/Period/InterestRate') AS x(x)

)

SELECT

code AS Code,

name AS Name,

MAX(Rate) AS HighestRateEver

FROM

RateData

GROUP BY

code, name;

**Explanation of the SQL Query:**

1. **Common Table Expression (CTE) RateData:**
   * **SELECT Clause:** We select Bank.code, Bank.name, and extract the Rate attribute from the XML.
   * **FROM Clause:** We join Bank and AccountType on the bank code.
   * **CROSS APPLY Clause:** We use the nodes method to shred the XML data in rates, creating a row for each InterestRate element.
   * **CAST Function:** We cast the extracted Rate attribute to a FLOAT for numerical comparison.
2. **Main Query:**
   * **SELECT Clause:** We select code (as Code), name (as Name), and the maximum rate (MAX(Rate)) for each bank.
   * **GROUP BY Clause:** We group the results by code and name to ensure one row per bank with the highest interest rate.

This query effectively parses the XML data, extracts the necessary information, and computes the highest interest rate per bank, fulfilling the requirements of the question.

A screenshot of a computer

Description automatically generated

**Solution 1: Using XMLQUERY and XMLAGG**

sql

Copy code

SELECT Code, Name,

(SELECT XMLQUERY('max($a//@Rate)'

PASSING XMLAGG(rates) AS "a")

FROM AccountType

WHERE bank = code) AS HighestRateEver

FROM Bank;

**Explanation:**

1. **SELECT Clause:** The main query selects Code and Name from the Bank table.
2. **Subquery:** For each bank, a subquery is used to find the maximum interest rate.
   * **XMLQUERY:** This function is used to execute an XQuery expression.
   * **'max($a//@Rate)':** This XQuery expression finds the maximum value of the Rate attribute within the provided XML data.
   * **PASSING XMLAGG(rates) AS "a":** The XMLAGG function aggregates the rates XML data from all AccountType rows related to the current bank and passes it to the XQuery expression as $a.
   * **WHERE bank = code:** This filters the AccountType rows to include only those that match the current bank's code.

**Solution 2: Using XMLTABLE and LEFT OUTER JOINs**

SELECT Code, Bank.Name, MAX(rate) AS HighestRateEver

FROM Bank

LEFT OUTER JOIN AccountType ON bank = code

LEFT OUTER JOIN XMLTABLE('$a//@Rate' PASSING rates AS "a"

COLUMNS rate REAL PATH '.') ON 1=1

GROUP BY Code, Bank.Name;

**Explanation:**

1. **LEFT OUTER JOIN AccountType:** This joins the Bank table with the AccountType table on the code and bank columns.
2. **LEFT OUTER JOIN XMLTABLE:** This joins the result with an XMLTABLE function.
   * **'$a//@Rate' PASSING rates AS "a":** This specifies the XQuery expression to extract the Rate attribute from the XML data, passing the rates XML column as $a.
   * **COLUMNS rate REAL PATH '.':** This defines the rate column to be of type REAL, extracting the value at the specified path.
   * **ON 1=1:** This join condition ensures that every row from the previous join is included.
3. **GROUP BY Clause:** Groups the results by Code and Bank.Name.
4. **MAX(rate):** This calculates the maximum rate for each group.

**Solution 3: Using XMLCAST and XMLQUERY**

SELECT Code, Bank.Name,

MAX(XMLCAST(XMLQUERY('max($a//@Rate)' PASSING rates AS "a") AS REAL)) AS HighestRateEver

FROM Bank

LEFT OUTER JOIN AccountType ON bank = code

GROUP BY Code, Bank.Name;

**Explanation:**

1. **LEFT OUTER JOIN AccountType:** This joins the Bank table with the AccountType table on the code and bank columns.
2. **MAX with XMLCAST and XMLQUERY:**
   * **XMLQUERY:** Executes the XQuery expression 'max($a//@Rate)' to find the maximum Rate attribute.
   * **PASSING rates AS "a":** Passes the rates XML column as $a.
   * **XMLCAST(... AS REAL):** Casts the result of the XQuery expression to a REAL type.
3. **GROUP BY Clause:** Groups the results by Code and Bank.Name.
4. **MAX(...):** This calculates the maximum rate for each group.

**Additional Notes:**

* **COALESCE Function:** The comment suggests that if you want to replace NULL values with 0, you can use the COALESCE function. For example:

COALESCE(MAX(rate), 0) AS HighestRateEver

* **Handling Banks Without Account Types:** The solutions using LEFT OUTER JOIN ensure that banks without any account types are included in the result set, potentially with a NULL or 0 value for HighestRateEver.

These queries provide different approaches to achieve the same goal, depending on how you want to handle XML data and join operations in your SQL environment.

A screenshot of a computer

Description automatically generated

**Explanation of the Provided SQL Query**

The query is designed to retrieve information about all account types that offer free withdrawals and structure the result in the specified XML format.

**Query Breakdown**

SELECT XMLFOREST(

XMLAGG(

XMLELEMENT(

NAME "Bank",

XMLATTRIBUTES(code AS "Code", name AS "Name"),

typesXML

)

) AS "Result"

)

FROM (

SELECT Code, Bank.Name,

XMLAGG(

XMLELEMENT(

NAME "AccountType",

XMLATTRIBUTES(AccountType.Name AS "Name")

)

) AS typesXML

FROM Bank

INNER JOIN AccountType ON bank = code

WHERE freeWithdrawals

GROUP BY Code, Bank.Name

);

**Explanation of Each Part**

1. **Main Query:**

sql

Copy code

SELECT XMLFOREST(

XMLAGG(

XMLELEMENT(

NAME "Bank",

XMLATTRIBUTES(code AS "Code", name AS "Name"),

typesXML

)

) AS "Result"

)

* + XMLFOREST: Creates a forest of XML elements, here used to wrap the result in a <Result> tag.
  + XMLAGG: Aggregates multiple XML elements into a single XML value, here used to gather multiple <Bank> elements.
  + XMLELEMENT: Creates an XML element.
    - NAME "Bank": Defines the name of the XML element as <Bank>.
    - XMLATTRIBUTES(code AS "Code", name AS "Name"): Adds Code and Name attributes to the <Bank> element.
    - typesXML: Includes the nested <AccountType> elements.

1. **Subquery:**

sql

Copy code

FROM (

SELECT Code, Bank.Name,

XMLAGG(

XMLELEMENT(

NAME "AccountType",

XMLATTRIBUTES(AccountType.Name AS "Name")

)

) AS typesXML

FROM Bank

INNER JOIN AccountType ON bank = code

WHERE freeWithdrawals

GROUP BY Code, Bank.Name

)

* + This subquery selects Code and Name from the Bank table.
  + XMLAGG is used to aggregate the nested <AccountType> elements.
  + XMLELEMENT creates the <AccountType> elements.
    - NAME "AccountType": Defines the name of the XML element as <AccountType>.
    - XMLATTRIBUTES(AccountType.Name AS "Name"): Adds the Name attribute to the <AccountType> element.

1. **INNER JOIN and Filtering:**

sql

Copy code

INNER JOIN AccountType ON bank = code

WHERE freeWithdrawals

GROUP BY Code, Bank.Name

* + Joins the Bank and AccountType tables on the bank and code columns.
  + Filters the AccountType rows where freeWithdrawals is TRUE.
  + Groups by Code and Bank.Name to ensure the correct aggregation.

**Final Output Structure**

The query ensures that the output XML is structured as follows:

<Result>

<Bank Name="" Code="">

<AccountType Name=""/>

</Bank>

<Bank Name="" Code="">

<AccountType Name=""/>

<AccountType Name=""/>

</Bank>

</Result>

* Each <Result> element contains multiple <Bank> elements.
* Each <Bank> element contains the Name and Code attributes.
* Each <Bank> element contains one or more <AccountType> elements with the Name attribute.

This structure matches the specified format and correctly reflects the information about account types that offer free withdrawals.

SELECT XMLFOREST (XMLAGG(XMLELEMENT(NAME “Bank”,

XMLATTRIBUTES(code AS “Code”, name AS “Name”),typesXML AS “Result”)

FROM ( SELECT Code,Bank.Name,

XMLAGG(XMLELEMENT(

A white background with black text

Description automatically generated

Which account types offer currently atleast 4% interest rate

if you have balance of 50,000.

SELECT XMLFOREST(

XMLAGG(XMLELEMENT(NAME “AccountType”,XMLATTRIBUTES(AccountType.bank AS “bankcode”, Bank.name AS “bankname” , AccountType.name AS “name” )) AS “Result”)

FROM Bank INNER JOIN AccountType ON bank=code

WHERE AccountType/InterestRate/@Rate=>4 AND AccountType/InterestRate/@MinimumBalance>50000

Cannot use path expressions directly in where clause without XMLEXISTS

The query should use XMLEXISTS to filter rows based on XML content.

**Corrected Query:**

To correct the query to properly handle the XML data and fulfill the requirements, it should use the appropriate XML functions as shown in the example from the image. The query from the image uses XMLEXISTS to filter for XML data that meets the criteria.

SELECT XMLFOREST(

XMLAGG(XMLELEMENT(NAME “AccountType”,XMLATTRIBUTES(AccountType.bank AS “bankcode”, Bank.name AS “bankname” , AccountType.name AS “name” )) AS “Result”)

FROM Bank INNER JOIN AccountType ON bank=code

WHERE XMLEXISTS ( ‘$a//Period[ not @EndDate] /InterestRate[@MinimumBalance<=50000 and @Rate>=4]’ PASSING AccountType.rates AS “a” )

SQL Server

A close-up of a text

Description automatically generated

Retrieve information about each bank according to the following structure.

<Result>

<Bank code=”” name=”” numberofActiveAccountTypes=””/>

SELECT XMLFOREST(XMLAGG(XMLELEMENT(NAME “Bank”, XMLATTRIBUTES

(Bank.code AS “code”,Bank.name AS “name”, SELECT XMLQUERY( COUNT(

SELECT code,name,(SELECT COUNT(\*)

FROM AccountType

WHERE bank=code

AND rates.exist(‘//Period[ not @EndDate]’)=1)

AS numberofActiveAccountTypes

FROM Bank

FOR XML RAW(‘Bank’),ROOT(‘Result’)