Prüfungsnummer:70-761-deutsch

Prüfungsname: Querying Data with Transact-SQL

Version:demo

1. Hinweis: Diese Aufgabe gehört zu einer Reihe von Fragestellungen, für die dieselben Antwortmöglichkeiten zur Auswahl stehen. Eine Antwort kann für mehr als eine Frage der Serie richtig sein. Die Fragen sind voneinander unabhängig. Die bereitgestellten Informationen und Details beziehen sich jeweils nur auf die Aufgabe, die diese Informationen enthält.
Sie führen die folgende Transact-SQL Anweisung aus, um eine neue Tabelle zu erstellen:

```
CREATE TABLE Customers (
    CustomerID int NOT NULL PRIMARY KEY CLUSTERED,
    FirstName nvarchar(100) NOT NULL,
    LastName nvarchar(100) NOT NULL,
    TaxIdNumber varchar(20) NOT NULL,
    Address nvarchar(1024) NOT NULL,
    ANNUALRevenue decimal(19,2) NOT NULL,
    DateCreated datetime2(2) NOT NULL,
    ValidFrom datetime2(2) GENERATED ALWAYS AS ROW START NOT NULL,
    ValidTo datetime2(2) GENERATED ALWAYS AS ROW END NOT NULL,
    PERIOD FOR SYSTEM_TIME(ValidFrom, ValidTo)
)
WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.CustomersHistory))
```

Sie müssen alle Kundendaten überprüfen.
Welche Transact-SQL Anweisung führen Sie aus?
(Im Hilfetext finden Sie zusätzliche Antwortmöglichkeiten.)

A.
```
SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue, DateCreated
FROM Customers
GROUP BY GROUPING SETS((FirstName, LastName), (Address), (CustomerID, AnnualRevenue), (CustomerID), ())
ORDER BY CustomerID, FirstName, LastName, Address, AnnualRevenue
```

B.
```
SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue, DateCreated, ValidFrom, ValidTo
FROM Customers
FOR SYSTEM_TIME ALL Order BY ValidFrom
```

C.
```
SELECT c.CustomerID, c.FirstName, c.LastName, c.Address, c.ValidFrom, c.ValidTo
FROM Customers AS c
ORDER BY c.CustomerID
FOR JSON AUTO, ROOT('Customers')
```

D.
```
SELECT * FROM (SELECT CustomerID, FirstName, LastName, Address, AnnualRevenue, DateCreated
FROM Customers) AS Customers PIVOT(AVG(AnnualRevenue)
```

FOR DateCreated IN ([2017])) AS PivotCustomers
ORDER BY LastName, FirstName
E.SELECT CustomerID, AVG(AnnualRevenue)
AS AverageAnnualRevenue, FirstName, LastName, Address, DateCreated
FROM Customers WHERE YEAR(DateCreated) >= 2017
GROUP BY CustomerID, FirstName, LastName, Address, DateCreated
F.SELECT c.CustomerID, c.FirstName, c.LastName, c.Address, c.ValidFrom, c.ValidTo
FROM Customers AS c
ORDER BY c.CustomerID
FOR XML PATH ('CustomerData'), ROOT('Customers')

Korrekte Antwort: B

Erläuterungen:
SQL Server 2016 führt die Unterstützung für temporale Tabellen mit Versionsverwaltung durch das System als Datenbankfeature ein, das integrierte Unterstützung für das Bereitstellen von Informationen zu den zu jedem Zeitpunkt in der Tabelle gespeicherten Daten mit sich bringt, anstatt nur die aktuell in einer Tabelle gespeicherten Daten zu unterstützen. Temporal ist ein Datenbankfeature, das mit ANSI SQL 2011 eingeführt wurde und jetzt in SQL Server 2016unterstützt wird.
Eine temporale Tabelle mit Versionsverwaltung durch das System ist ein neuer Benutzertabellentyp in SQL Server 2016, der dafür ausgelegt ist, den Verlauf aller Datenänderungen lückenlos zu speichern und einfache Zeitpunktanalysen zu ermöglichen. Bei diesem Typ von temporaler Tabelle spricht man von Versionsverwaltung durch das System, da die Gültigkeitsdauer für jede Zeile vom System (d.h. vom Datenbankmodul) verwaltet wird.
Jede temporale Tabelle weist zwei explizit definierte Spalten auf, beide vom Datentyp datetime2 . Diese Spalten werden als Zeitraumspalten bezeichnet. Diese Zeitraumspalten werden bei jeder Änderung einer Zeile ausschließlich vom System zum Aufzeichnen des Gültigkeitszeitraums verwendet.
Über diese Zeitraumspalten hinaus enthält eine temporale Tabelle außerdem einen Verweis auf eine weitere Tabelle mit einem gespiegelten Schema. Das System verwendet diese Tabelle, um bei jeder Aktualisierung oder Löschung einer Zeile in der temporalen Tabelle automatisch die Vorversion der Zeile zu speichern. Diese zusätzliche Tabelle wird als die Verlaufstabelle bezeichnet, während die Haupttabelle, die die aktuellen (gültigen) Zeilenversionen speichert, als die aktuelle Tabelle oder einfach als die temporale Tabelle bezeichnet wird. Während der Erstellung von temporalen Tabellen können Benutzer eine vorhandene Verlaufstabelle (deren Schema kompatibel sein muss) angeben oder vom System eine standardmäßige Verlaufstabelle erstellen lassen.
Der Ausdruck ALL gibt die Vereinigungsmenge der Zeilen zurück, die der aktuellen und der Verlaufstabelle angehören.
Erweiterte Antwortmöglichkeiten:
Hätten Sie es auch gewusst, wenn mehr als die gezeigten 6 Antwortmöglichkeiten zur Auswahl stünden?
A:

SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue, DateCreated
FROM Customers
GROUP BY GROUPING SETS (FirstName, LastName), Address, CustomerID, AnnualRevenue), CustomerID, ())
ORDER BY CustomerID, FirstName, LastName, Address, AnnualRevenue
B:

SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue, DateCreated, ValidFrom, ValidTo
FROM Customers
FOR SYSTEM_TIME ALL Order BY ValidFrom
C:

SELECT c.CustomerID, c.FirstName, c.LastName, c.Address, c.ValidFrom, c.ValidTo
FROM Customers AS c
ORDER BY c.CustomerID
FOR JSON AUTO, ROOT('Customers')
D:

SELECT * FROM (SELECT CustomerID, FirstName, LastName, Address, AnnualRevenue, DateCreated
FROM Customers) AS Customers PIVOT(AVG(AnnualRevenue)
FOR DateCreated IN ([2017])) AS PivotCustomers
ORDER BY LastName, FirstName
E:

SELECT CustomerID, AVG(AnnualRevenue)
AS AverageAnnualRevenue, FirstName, LastName, Address, DateCreated
FROM Customers WHERE YEAR(DateCreated) >= 2017
GROUP BY CustomerID, FirstName, LastName, Address, DateCreated
F:

SELECT c.CustomerID, c.FirstName, c.LastName, c.Address, c.ValidFrom, c.ValidTo
FROM Customers AS c
ORDER BY c.CustomerID
FOR XML PATH ('CustomerData'), ROOT('Customers')
G:

SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, ValidFrom, ValidTo
FROM Customers
FOR SYSTEM_TIME BETWEEN '2017-01-01 00:00:00.000000' AND '2018-01-01 00:00:00.000000'

H:

SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, ValidFrom, ValidTo
FROM Customers
WHERE DateCreated BETWEEN '20170101' AND '20171231'

2. Hinweis: Diese Aufgabe gehört zu einer Reihe von Fragestellungen, für die dieselben Antwortmöglichkeiten zur Auswahl stehen. Eine Antwort kann für mehr als eine Frage der Serie richtig sein. Die Fragen sind voneinander unabhängig. Die bereitgestellten Informationen und Details beziehen sich jeweils nur auf die Aufgabe, die diese Informationen enthält.
Sie führen die folgende Transact-SQL Anweisung aus, um eine neue Tabelle zu erstellen:

```
CREATE TABLE Customers (
    CustomerID int NOT NULL PRIMARY KEY CLUSTERED,
    FirstName nvarchar(100) NOT NULL,
    LastName nvarchar(100) NOT NULL,
    TaxIdNumber varchar(20) NOT NULL,
    Address nvarchar(1024) NOT NULL,
    ANNUALRevenue decimal(19,2) NOT NULL,
    DateCreated datetime2(2) NOT NULL,
    ValidFrom datetime2(2) GENERATED ALWAYS AS ROW START NOT NULL,
    ValidTo datetime2(2) GENERATED ALWAYS AS ROW END NOT NULL,
    PERIOD FOR SYSTEM_TIME(ValidFrom, ValidTo)
)
WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.CustomersHistory))
```

Die Daten, die in der Tabelle gespeichert sind, sollen mithilfe von AJAX-Aufrufen und REST-Endpunkten zwischen Webseiten und Webservern übertragen werden.
Sie müssen alle Kundendaten in einem Format abfragen, das textbasiert ist und einen geringen Overhead hat.
Welche Transact-SQL Anweisung führen Sie aus?
(Im Hilfetext finden Sie zusätzliche Antwortmöglichkeiten.)

A.SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue, DateCreated
FROM Customers
GROUP BY GROUPING SETS((FirstName, LastName), (Address), (CustomerID, AnnualRevenue), (CustomerID), ())
ORDER BY CustomerID, FirstName, LastName, Address, AnnualRevenue
B.SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue, DateCreated, ValidFrom, ValidTo
FROM Customers
FOR SYSTEM_TIME ALL Order BY ValidFrom
C.SELECT c.CustomerID, c.FirstName, c.LastName, c.Address, c.ValidFrom, c.ValidTo
FROM Customers AS c

ORDER BY c.CustomerID
FOR JSON AUTO, ROOT('Customers')
D.SELECT * FROM (SELECT CustomerID, FirstName, LastName, Address, AnnualRevenue, DateCreated
FROM Customers) AS Customers PIVOT(AVG(AnnualRevenue)
FOR DateCreated IN ([2017])) AS PivotCustomers
ORDER BY LastName, FirstName
E.SELECT CustomerID, AVG(AnnualRevenue)
AS AverageAnnualRevenue, FirstName, LastName, Address, DateCreated
FROM Customers WHERE YEAR(DateCreated) >= 2017
GROUP BY CustomerID, FirstName, LastName, Address, DateCreated
F.SELECT c.CustomerID, c.FirstName, c.LastName, c.Address, c.ValidFrom, c.ValidTo
FROM Customers AS c
ORDER BY c.CustomerID
FOR XML PATH ('CustomerData'), ROOT('Customers')

Korrekte Antwort: C

Erläuterungen:
Die Möglichkeit, Abfrageergebnisse als JSON zu formatieren oder zu exportieren, indem Sie die FOR JSON-Klausel einer SELECT-Anweisung hinzufügen ist eine der Neuerungen von SQL Server 2016. Verwenden der FOR JSON-Klausel, um Clientanwendungen zu vereinfachen, indem Sie die Formatierung der JSON-Ausgabe von der App zu SQL Server delegieren.
Bei Verwendung der FOR JSON -Klausel, können Sie die Struktur der JSON-Ausgabe explizit angeben, oder lassen Sie die Struktur der SELECT-Anweisung die Ausgabe bestimmen.

Verwenden Sie FOR JSON PATH, um die vollständige Kontrolle über das Format der JSON-Ausgabe zu behalten. Sie können Wrapper-Objekte erstellen und komplexe Eigenschaften schachteln.

Verwenden Sie FOR JSON AUTO, um die JSON-Ausgabe automatisch auf Grundlage der SELECT-Anweisung zu formatieren.
Erweiterte Antwortmöglichkeiten:
Hätten Sie es auch gewußt, wenn mehr als die gezeigten 6 Antwortmöglichkeiten zur Auswahl stünden?
A:

SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue, DateCreated
FROM Customers
GROUP BY GROUPING SETS (FirstName, LastName), Address, CustomerID, AnnualRevenue), CustomerID, ())
ORDER BY CustomerID, FirstName, LastName, Address, AnnualRevenue
B:

SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue,

DateCreated, ValidFrom, ValidTo
FROM Customers
FOR SYSTEM_TIME ALL Order BY ValidFrom
C:

SELECT c.CustomerID, c.FirstName, c.LastName, c.Address, c.ValidFrom, c.ValidTo
FROM Customers AS c
ORDER BY c.CustomerID
FOR JSON AUTO, ROOT('Customers')
D:

SELECT * FROM (SELECT CustomerID, FirstName, LastName, Address, AnnualRevenue, DateCreated
FROM Customers) AS Customers PIVOT(AVG(AnnualRevenue)
FOR DateCreated IN ([2017])) AS PivotCustomers
ORDER BY LastName, FirstName
E:

SELECT CustomerID, AVG(AnnualRevenue)
AS AverageAnnualRevenue, FirstName, LastName, Address, DateCreated
FROM Customers WHERE YEAR(DateCreated) >= 2017
GROUP BY CustomerID, FirstName, LastName, Address, DateCreated
F:

SELECT c.CustomerID, c.FirstName, c.LastName, c.Address, c.ValidFrom, c.ValidTo
FROM Customers AS c
ORDER BY c.CustomerID
FOR XML PATH ('CustomerData'), ROOT('Customers')
G:

SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, ValidFrom, ValidTo
FROM Customers
FOR SYSTEM_TIME BETWEEN '2017-01-01 00:00:00.000000' AND '2018-01-01 00:00:00.000000'
H:

SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, ValidFrom, ValidTo
FROM Customers
WHERE DateCreated BETWEEN '20170101' AND '20171231'

3. Hinweis: Diese Aufgabe gehört zu einer Reihe von Fragestellungen, für die dieselben Antwortmöglichkeiten zur Auswahl stehen. Eine Antwort kann für mehr als eine Frage der Serie richtig sein. Die Fragen sind voneinander unabhängig. Die bereitgestellten Informationen und

Details beziehen sich jeweils nur auf die Aufgabe, die diese Informationen enthält.
Sie führen die folgende Transact-SQL Anweisung aus, um eine neue Tabelle zu erstellen:

```
CREATE TABLE Customers (
    CustomerID int NOT NULL PRIMARY KEY CLUSTERED,
    FirstName nvarchar(100) NOT NULL,
    LastName nvarchar(100) NOT NULL,
    TaxIdNumber varchar(20) NOT NULL,
    Address nvarchar(1024) NOT NULL,
    ANNUALRevenue decimal(19,2) NOT NULL,
    DateCreated datetime2(2) NOT NULL,
    ValidFrom datetime2(2) GENERATED ALWAYS AS ROW START NOT NULL,
    ValidTo datetime2(2) GENERATED ALWAYS AS ROW END NOT NULL,
    PERIOD FOR SYSTEM_TIME(ValidFrom, ValidTo)
)
WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.CustomersHistory))
```

Sie entwickeln einen Bericht, der Kundendaten anzeigt. Der Bericht muss eine Spalte mit einer Gesamtsumme enthalten.
Sie müssen eine Abfrage schreiben, die die Daten für den Bericht abruft.
Welche Transact-SQL Anweisung führen Sie aus?
(Im Hilfetext finden Sie zusätzliche Antwortmöglichkeiten.)

A.
```
SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue, DateCreated
FROM Customers
GROUP BY GROUPING SETS((FirstName, LastName), (Address), (CustomerID, AnnualRevenue), (CustomerID), ())
ORDER BY CustomerID, FirstName, LastName, Address, AnnualRevenue
```

B.
```
SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue, DateCreated, ValidFrom, ValidTo
FROM Customers
FOR SYSTEM_TIME ALL Order BY ValidFrom
```

C.
```
SELECT c.CustomerID, c.FirstName, c.LastName, c.Address, c.ValidFrom, c.ValidTo
FROM Customers AS c
ORDER BY c.CustomerID
FOR JSON AUTO, ROOT('Customers')
```

D.
```
SELECT * FROM (SELECT CustomerID, FirstName, LastName, Address, AnnualRevenue, DateCreated
FROM Customers) AS Customers PIVOT(AVG(AnnualRevenue)
FOR DateCreated IN ([2017])) AS PivotCustomers
ORDER BY LastName, FirstName
```

E.
```
SELECT CustomerID, AVG(AnnualRevenue)
AS AverageAnnualRevenue, FirstName, LastName, Address, DateCreated
FROM Customers WHERE YEAR(DateCreated) >= 2017
GROUP BY CustomerID, FirstName, LastName, Address, DateCreated
```

F.SELECT c.CustomerID, c.FirstName, c.LastName, c.Address, c.ValidFrom, c.ValidTo
FROM Customers AS c
ORDER BY c.CustomerID
FOR XML PATH ('CustomerData'), ROOT('Customers')

Korrekte Antwort: E

Erläuterungen:
Antwort E ist die einzige Lösung in der eine Aggregatfunktion verwendet und Werte gruppiert werden. Die Lösung erscheint als beste Wahl für die Erstellung des Berichts.
Erweiterte Antwortmöglichkeiten:
Hätten Sie es auch gewußt, wenn mehr als die gezeigten 6 Antwortmöglichkeiten zur Auswahl stünden?
A:

SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue, DateCreated
FROM Customers
GROUP BY GROUPING SETS (FirstName, LastName), Address, CustomerID, AnnualRevenue), CustomerID, ())
ORDER BY CustomerID, FirstName, LastName, Address, AnnualRevenue
B:

SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue, DateCreated, ValidFrom, ValidTo
FROM Customers
FOR SYSTEM_TIME ALL Order BY ValidFrom
C:

SELECT c.CustomerID, c.FirstName, c.LastName, c.Address, c.ValidFrom, c.ValidTo
FROM Customers AS c
ORDER BY c.CustomerID
FOR JSON AUTO, ROOT('Customers')
D:

SELECT * FROM (SELECT CustomerID, FirstName, LastName, Address, AnnualRevenue, DateCreated
FROM Customers) AS Customers PIVOT(AVG(AnnualRevenue)
FOR DateCreated IN ([2017])) AS PivotCustomers
ORDER BY LastName, FirstName
E:

SELECT CustomerID, AVG(AnnualRevenue)
AS AverageAnnualRevenue, FirstName, LastName, Address, DateCreated

FROM Customers WHERE YEAR(DateCreated) >= 2017
GROUP BY CustomerID, FirstName, LastName, Address, DateCreated
F:

SELECT c.CustomerID, c.FirstName, c.LastName, c.Address, c.ValidFrom, c.ValidTo
FROM Customers AS c
ORDER BY c.CustomerID
FOR XML PATH ('CustomerData'), ROOT('Customers')
G:

SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, ValidFrom, ValidTo
FROM Customers
FOR SYSTEM_TIME BETWEEN '2017-01-01 00:00:00.000000' AND '2018-01-01 00:00:00.000000'
H:

SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, ValidFrom, ValidTo
FROM Customers
WHERE DateCreated BETWEEN '20170101' AND '20171231'

4. Hinweis: Diese Aufgabe gehört zu einer Reihe von Fragestellungen, für die dieselben Antwortmöglichkeiten zur Auswahl stehen. Eine Antwort kann für mehr als eine Frage der Serie richtig sein. Die Fragen sind voneinander unabhängig. Die bereitgestellten Informationen und Details beziehen sich jeweils nur auf die Aufgabe, die diese Informationen enthält.
Sie führen die folgende Transact-SQL Anweisung aus, um eine neue Tabelle zu erstellen:

```
CREATE TABLE Customers (
    CustomerID int NOT NULL PRIMARY KEY CLUSTERED,
    FirstName nvarchar(100) NOT NULL,
    LastName nvarchar(100) NOT NULL,
    TaxIdNumber varchar(20) NOT NULL,
    Address nvarchar(1024) NOT NULL,
    ANNUALRevenue decimal(19,2) NOT NULL,
    DateCreated datetime2(2) NOT NULL,
    ValidFrom datetime2(2) GENERATED ALWAYS AS ROW START NOT NULL,
    ValidTo datetime2(2) GENERATED ALWAYS AS ROW END NOT NULL,
    PERIOD FOR SYSTEM_TIME(ValidFrom, ValidTo)
)
WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.CustomersHistory))
```

Sie entwickeln einen Bericht, der aggregierte Kundendaten für das Jahr 2017 anzeigt. Der Bericht erfordert, dass die Daten denormalisiert vorliegen.
Sie müssen eine Abfrage für das Abrufen der Daten für den Bericht schreiben.
Welche Transact-SQL Anweisung führen Sie aus?

(Im Hilfetext finden Sie zusätzliche Antwortmöglichkeiten.)

A.SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue, DateCreated
FROM Customers
GROUP BY GROUPING SETS((FirstName, LastName), (Address), (CustomerID, AnnualRevenue), (CustomerID), ())
ORDER BY CustomerID, FirstName, LastName, Address, AnnualRevenue
B.SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue, DateCreated, ValidFrom, ValidTo
FROM Customers
FOR SYSTEM_TIME ALL Order BY ValidFrom
C.SELECT c.CustomerID, c.FirstName, c.LastName, c.Address, c.ValidFrom, c.ValidTo
FROM Customers AS c
ORDER BY c.CustomerID
FOR JSON AUTO, ROOT('Customers')
D.SELECT * FROM (SELECT CustomerID, FirstName, LastName, Address, AnnualRevenue, DateCreated
FROM Customers) AS Customers PIVOT(AVG(AnnualRevenue)
FOR DateCreated IN ([2017])) AS PivotCustomers
ORDER BY LastName, FirstName
E.SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, ValidFrom, ValidTo
FROM Customers
FOR SYSTEM_TIME BETWEEN '2017-01-01 00:00:00.000000' AND '2018-01-01 00:00:00.000000'
F.SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, ValidFrom, ValidTo
FROM Customers
WHERE DateCreated BETWEEN '20170101' AND '20171231'

Korrekte Antwort: E

Erläuterungen:
In Frage kommen die Lösungen der Antworten E und F. Während Antwort E auch die Daten aus der Verlaufstabelle in den Ergebnissatz einbezieht, liefert die Lösung aus Antwort F nur die aktuellen Daten. In der Aufgabe heißt es, dass die Daten denormalisiert benötigt werden. Dies lässt darauf schließen, dass die historischen Daten aus der Verlaufstabelle in das Ergebnis einbezogen werden sollen.
Erweiterte Antwortmöglichkeiten:
Hätten Sie es auch gewußt, wenn mehr als die gezeigten 6 Antwortmöglichkeiten zur Auswahl stünden?
A:

SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue, DateCreated

FROM Customers
GROUP BY GROUPING SETS (FirstName, LastName), Address, CustomerID, AnnualRevenue), CustomerID, ())
ORDER BY CustomerID, FirstName, LastName, Address, AnnualRevenue
B:

SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue, DateCreated, ValidFrom, ValidTo
FROM Customers
FOR SYSTEM_TIME ALL Order BY ValidFrom
C:

SELECT c.CustomerID, c.FirstName, c.LastName, c.Address, c.ValidFrom, c.ValidTo
FROM Customers AS c
ORDER BY c.CustomerID
FOR JSON AUTO, ROOT('Customers')
D:

SELECT * FROM (SELECT CustomerID, FirstName, LastName, Address, AnnualRevenue, DateCreated
FROM Customers) AS Customers PIVOT(AVG(AnnualRevenue)
FOR DateCreated IN ([2017])) AS PivotCustomers
ORDER BY LastName, FirstName
E:

SELECT CustomerID, AVG(AnnualRevenue)
AS AverageAnnualRevenue, FirstName, LastName, Address, DateCreated
FROM Customers WHERE YEAR(DateCreated) >= 2017
GROUP BY CustomerID, FirstName, LastName, Address, DateCreated
F:

SELECT c.CustomerID, c.FirstName, c.LastName, c.Address, c.ValidFrom, c.ValidTo
FROM Customers AS c
ORDER BY c.CustomerID
FOR XML PATH ('CustomerData'), ROOT('Customers')
G:

SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, ValidFrom, ValidTo
FROM Customers
FOR SYSTEM_TIME BETWEEN '2017-01-01 00:00:00.000000' AND '2018-01-01 00:00:00.000000'
H:

SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, ValidFrom, ValidTo

FROM Customers
WHERE DateCreated BETWEEN '20170101' AND '20171231'

5. Hinweis: Diese Aufgabe gehört zu einer Reihe von Fragestellungen, für die dieselben Antwortmöglichkeiten zur Auswahl stehen. Eine Antwort kann für mehr als eine Frage der Serie richtig sein. Die Fragen sind voneinander unabhängig. Die bereitgestellten Informationen und Details beziehen sich jeweils nur auf die Aufgabe, die diese Informationen enthält.
Sie führen die folgende Transact-SQL Anweisung aus, um eine neue Tabelle zu erstellen:

```
CREATE TABLE Customers (
    CustomerID int NOT NULL PRIMARY KEY CLUSTERED,
    FirstName nvarchar(100) NOT NULL,
    LastName nvarchar(100) NOT NULL,
    TaxIdNumber varchar(20) NOT NULL,
    Address nvarchar(1024) NOT NULL,
    ANNUALRevenue decimal(19,2) NOT NULL,
    DateCreated datetime2(2) NOT NULL,
    ValidFrom datetime2(2) GENERATED ALWAYS AS ROW START NOT NULL,
    ValidTo datetime2(2) GENERATED ALWAYS AS ROW END NOT NULL,
    PERIOD FOR SYSTEM_TIME(ValidFrom, ValidTo)
)
WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.CustomersHistory))
```

Sie müssen eine Abfrage entwickeln, die folgenden Anforderungen entspricht:
Daten sollen in einer baumartigen Struktur ausgegeben werden.
Es sollen gemischte Inhaltstypen unterstützt werden.
Es sollen benutzerdefinierte Metadaten-Attribute verwendet werden.
Welche Transact-SQL Anweisung führen Sie aus?
(Im Hilfetext finden Sie zusätzliche Antwortmöglichkeiten.)

A.SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue, DateCreated
FROM Customers
GROUP BY GROUPING SETS((FirstName, LastName), (Address), (CustomerID, AnnualRevenue), (CustomerID), ())
ORDER BY CustomerID, FirstName, LastName, Address, AnnualRevenue
B.SELECT FirstName, LastName, Address
FROM Customers
FOR SYSTEM_TIME ALL Order BY ValidFrom
C.SELECT c.CustomerID, c.FirstName, c.LastName, c.Address, c.ValidFrom, c.ValidTo
FROM Customers AS c
ORDER BY c.CustomerID
FOR JSON AUTO, ROOT('Customers')
D.SELECT * FROM (SELECT CustomerID, FirstName, LastName, Address, AnnualRevenue, DateCreated

FROM Customers) AS Customers PIVOT(AVG(AnnualRevenue)
FOR DateCreated IN ([2017])) AS PivotCustomers
ORDER BY LastName, FirstName
E.SELECT CustomerID, AVG(AnnualRevenue)
AS AverageAnnualRevenue, FirstName, LastName, Address, DateCreated
FROM Customers WHERE YEAR(DateCreated) >= 2017
GROUP BY CustomerID, FirstName, LastName, Address, DateCreated
F.SELECT c.CustomerID, c.FirstName, c.LastName, c.Address, c.ValidFrom, c.ValidTo
FROM Customers AS c
ORDER BY c.CustomerID
FOR XML PATH ('CustomerData'), ROOT('Customers')

Korrekte Antwort: F

Erläuterungen:
Eine SELECT-Abfrage gibt Ergebnisse als Rowset zurück. Sie können optional die formalen Ergebnisse einer SQL-Abfrage als XML abrufen, indem Sie die FOR XML-Klausel in der Abfrage angeben. Die FOR XML-Klausel kann in Abfragen der obersten Ebene sowie in Unterabfragen verwendet werden. Die FOR XML-Klausel der obersten Ebene kann nur in der SELECT-Anweisung verwendet werden. In Unterabfragen kann FOR XML in den INSERT-, UPDATE- und DELETE-Anweisungen verwendet werden. Die Klausel kann auch in Zuweisungsanweisungen verwendet werden.
In einer FOR XML-Klausel geben Sie einen der folgenden Modi an:

RAW

AUTO

EXPLICIT

PATH

Der RAW-Modus generiert ein einzelnes <row>-Element pro Zeile im Rowset, das von der SELECT-Anweisung zurückgegeben wird. Sie können eine XML-Hierarchie erstellen, indem Sie geschachtelte FOR XML-Abfragen schreiben.

Der AUTO-Modus generiert mithilfe von Heuristik, basierend auf der Art der Angabe der SELECT-Anweisung, Schachtelung in dem sich ergebenden XML. Sie besitzen nur minimale Kontrolle über die Form des generierten XML. Die geschachtelten FOR XML-Abfragen können geschrieben werden, um eine XML-Hierarchie zu erstellen, die über die XML-Form hinausgeht, die von der Heuristik des AUTO-Modus generiert wird.

Der EXPLICIT-Modus lässt eine größere Kontrolle über die Form des XML zu. Sie können Attribute und Elemente beliebig mischen, um die Form des XML zu bestimmen. Aufgrund der Abfrageausführung ist ein bestimmtes Format für das sich ergebende Rowset erforderlich, das generiert wird. Dieses Rowsetformat wird anschließend der XML-Form zugeordnet. Die Leistungsfähigkeit des EXPLICIT-Modus beruht auf der Möglichkeit, Attribute und Elemente beliebig zu mischen, Wrapper und verschachtelte komplexe Eigenschaften, durch Leerzeichen

getrennte Werte (das OrderID-Attribut kann z. B. eine Liste von Bestellungs-ID-Werten besitzen) und gemischte Inhalte erstellen zu können.

Das Schreiben von Abfragen für den EXPLICIT-Modus kann jedoch aufwendig sein. Sie können einige der neuen FOR XML-Funktionen verwenden; Sie können z. B. verschachtelte Abfragen für den FOR XML RAW/AUTO/PATH-Modus und die TYPE-Direktive schreiben, statt den EXPLICIT-Modus zum Generieren der Hierarchien zu verwenden. Die geschachtelten FOR XML-Abfragen können beliebiges XML erstellen, das mithilfe des EXPLICIT-Modus generiert werden kann.

Der PATH-Modus stellt zusammen mit der Möglichkeit geschachtelter FOR XML-Abfragen die Flexibilität des EXPLICIT-Modus auf einfachere Weise bereit.

Erweiterte Antwortmöglichkeiten:

Hätten Sie es auch gewußt, wenn mehr als die gezeigten 6 Antwortmöglichkeiten zur Auswahl stünden?

A:

```
SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue, DateCreated
FROM Customers
GROUP BY GROUPING SETS (FirstName, LastName), Address, CustomerID, AnnualRevenue), CustomerID, ())
ORDER BY CustomerID, FirstName, LastName, Address, AnnualRevenue
```

B:

```
SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue, DateCreated, ValidFrom, ValidTo
FROM Customers
FOR SYSTEM_TIME ALL Order BY ValidFrom
```

C:

```
SELECT c.CustomerID, c.FirstName, c.LastName, c.Address, c.ValidFrom, c.ValidTo
FROM Customers AS c
ORDER BY c.CustomerID
FOR JSON AUTO, ROOT('Customers')
```

D:

```
SELECT * FROM (SELECT CustomerID, FirstName, LastName, Address, AnnualRevenue, DateCreated
FROM Customers) AS Customers PIVOT(AVG(AnnualRevenue)
FOR DateCreated IN ([2017])) AS PivotCustomers
ORDER BY LastName, FirstName
```

E:

```
SELECT CustomerID, AVG(AnnualRevenue)
AS AverageAnnualRevenue, FirstName, LastName, Address, DateCreated
```

FROM Customers WHERE YEAR(DateCreated) >= 2017
GROUP BY CustomerID, FirstName, LastName, Address, DateCreated
F:

SELECT c.CustomerID, c.FirstName, c.LastName, c.Address, c.ValidFrom, c.ValidTo
FROM Customers AS c
ORDER BY c.CustomerID
FOR XML PATH ('CustomerData'), ROOT('Customers')
G:

SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, ValidFrom, ValidTo
FROM Customers
FOR SYSTEM_TIME BETWEEN '2017-01-01 00:00:00.000000' AND '2018-01-01 00:00:00.000000'
H:

SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, ValidFrom, ValidTo
FROM Customers
WHERE DateCreated BETWEEN '20170101' AND '20171231'

6. Hinweis: Diese Aufgabe gehört zu einer Reihe von Fragestellungen, für die dieselben Antwortmöglichkeiten zur Auswahl stehen. Eine Antwort kann für mehr als eine Frage der Serie richtig sein. Die Fragen sind voneinander unabhängig. Die bereitgestellten Informationen und Details beziehen sich jeweils nur auf die Aufgabe, die diese Informationen enthält.
Sie führen die folgende Transact-SQL Anweisung aus, um eine neue Tabelle zu erstellen:

```
CREATE TABLE Customers (
    CustomerID int NOT NULL PRIMARY KEY CLUSTERED,
    FirstName nvarchar(100) NOT NULL,
    LastName nvarchar(100) NOT NULL,
    TaxIdNumber varchar(20) NOT NULL,
    Address nvarchar(1024) NOT NULL,
    ANNUALRevenue decimal(19,2) NOT NULL,
    DateCreated datetime2(2) NOT NULL,
    ValidFrom datetime2(2) GENERATED ALWAYS AS ROW START NOT NULL,
    ValidTo datetime2(2) GENERATED ALWAYS AS ROW END NOT NULL,
    PERIOD FOR SYSTEM_TIME(ValidFrom, ValidTo)
)
WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.CustomersHistory))
```

Sie entwickeln einen Bericht, der aggregierte Kundendaten für das Jahr 2017 anzeigt. Der Bericht erfordert, dass die Daten denormalisiert vorliegen.
Sie müssen eine Abfrage für das Abrufen der Daten für den Bericht schreiben.
Welche Transact-SQL Anweisung führen Sie aus?
(Im Hilfetext finden Sie zusätzliche Antwortmöglichkeiten.)

A.SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue, DateCreated
FROM Customers
GROUP BY GROUPING SETS((FirstName, LastName), (Address), (CustomerID, AnnualRevenue), (CustomerID), ())
ORDER BY CustomerID, FirstName, LastName, Address, AnnualRevenue
B.SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue, DateCreated, ValidFrom, ValidTo
FROM Customers
FOR SYSTEM_TIME ALL Order BY ValidFrom
C.SELECT c.CustomerID, c.FirstName, c.LastName, c.Address, c.ValidFrom, c.ValidTo
FROM Customers AS c
ORDER BY c.CustomerID
FOR JSON AUTO, ROOT('Customers')
D.SELECT * FROM (SELECT CustomerID, FirstName, LastName, Address, AnnualRevenue, DateCreated
FROM Customers) AS Customers PIVOT(AVG(AnnualRevenue)
FOR DateCreated IN ([2017])) AS PivotCustomers
ORDER BY LastName, FirstName
E.SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, ValidFrom, ValidTo
FROM Customers
FOR SYSTEM_TIME BETWEEN '2017-01-01 00:00:00.000000' AND '2018-01-01 00:00:00.000000'
F.SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, ValidFrom, ValidTo
FROM Customers
WHERE DateCreated BETWEEN '20170101' AND '20171231'

Korrekte Antwort: F

Erläuterungen:
In Frage kommen die Lösungen der Antworten E und F. Während Antwort E auch die Daten aus der Verlaufstabelle in den Ergebnissatz einbezieht, liefert die Lösung aus Antwort F nur die aktuellen Daten. In der Aufgabe heißt es, dass die Daten ormalisiert benötigt werden. Dies lässt darauf schließen, dass die historischen Daten aus der Verlaufstabelle nicht in das Ergebnis einbezogen werden sollen.
Erweiterte Antwortmöglichkeiten:
Hätten Sie es auch gewußt, wenn mehr als die gezeigten 6 Antwortmöglichkeiten zur Auswahl stünden?
A:

SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue, DateCreated
FROM Customers

GROUP BY GROUPING SETS (FirstName, LastName), Address, CustomerID, AnnualRevenue), CustomerID, ())
ORDER BY CustomerID, FirstName, LastName, Address, AnnualRevenue
B:

SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, AnnualRevenue, DateCreated, ValidFrom, ValidTo
FROM Customers
FOR SYSTEM_TIME ALL Order BY ValidFrom
C:

SELECT c.CustomerID, c.FirstName, c.LastName, c.Address, c.ValidFrom, c.ValidTo
FROM Customers AS c
ORDER BY c.CustomerID
FOR JSON AUTO, ROOT('Customers')
D:

SELECT * FROM (SELECT CustomerID, FirstName, LastName, Address, AnnualRevenue, DateCreated
FROM Customers) AS Customers PIVOT(AVG(AnnualRevenue)
FOR DateCreated IN ([2017])) AS PivotCustomers
ORDER BY LastName, FirstName
E:

SELECT CustomerID, AVG(AnnualRevenue)
AS AverageAnnualRevenue, FirstName, LastName, Address, DateCreated
FROM Customers WHERE YEAR(DateCreated) >= 2017
GROUP BY CustomerID, FirstName, LastName, Address, DateCreated
F:

SELECT c.CustomerID, c.FirstName, c.LastName, c.Address, c.ValidFrom, c.ValidTo
FROM Customers AS c
ORDER BY c.CustomerID
FOR XML PATH ('CustomerData'), ROOT('Customers')
G:

SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, ValidFrom, ValidTo
FROM Customers
FOR SYSTEM_TIME BETWEEN '2017-01-01 00:00:00.000000' AND '2018-01-01 00:00:00.000000'
H:

SELECT CustomerID, FirstName, LastName, TaxIdNumber, Address, ValidFrom, ValidTo
FROM Customers

WHERE DateCreated BETWEEN '20170101' AND '20171231'

7. Transaktionen geben eine Isolationsstufe an, mit der definiert wird, bis zu welchem Ausmaß eine Transaktion von Ressourcen- oder Datenänderungen isoliert sein muss, die von anderen Transaktionen durchgeführt werden.
Welches ist die Standard-Isolationsstufe für Datenbanken einer SQL Server 2016-Instanz?

A.READ COMMITTED
B.READ UNCOMMITTED
C.REPEATABLE READ
D.SERIALIZABLE

Korrekte Antwort: A

Erläuterungen:
Sie können den Befehl DBCC UserOptions auf eine beliebige Datenbank anwenden, um Detailinformationen zu Spracheinstellungen, Datumsformat und Isolationsstufe abzurufen.
Nachstehend wird die beispielhafte Ausgabe des Befehls für eine Datenbank auf einer SQL Server 2016-Instanz gezeigt:

Set Option	Value
textsize	2147483647
language	Deutsch
dateformat	dmy
datefirst	1
lock_timeout	-1
quoted_identifier	SET
arithabort	SET
ansi_null_dflt_on	SET
ansi_warnings	SET
ansi_padding	SET
ansi_nulls	SET
concat_null_yields_null	SET
isolation level	read committed