microsoft.test-king.70-464.v2020-09-17.by.partyxpre.120q.vce

70-464

70-464

Score:	800/1000
Version:	1
Time Limit:	120 Minutes

Topic 1, Scenario 1

Case Study (5 questions)

Scenario 1

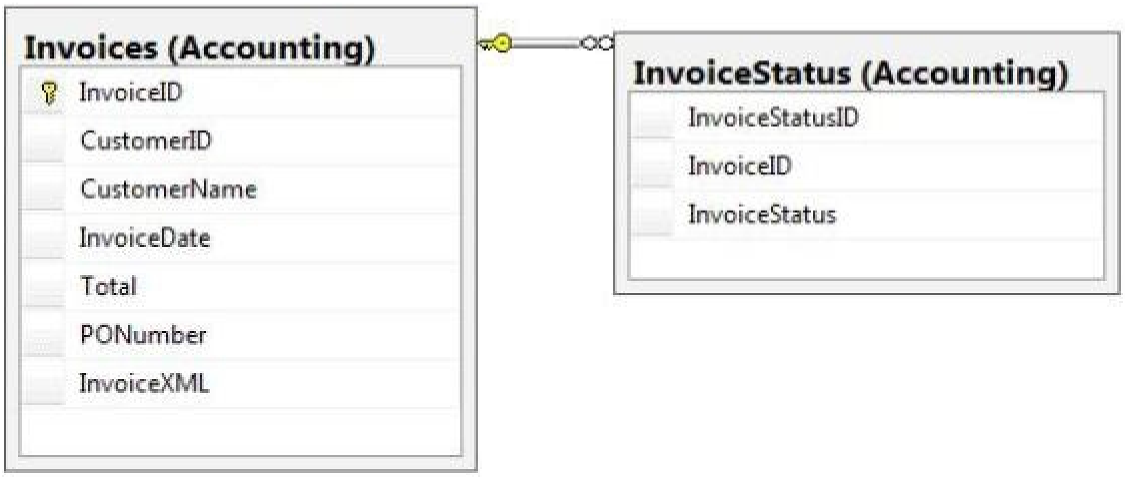
Application Information

Your company receives invoices in XML format from customers. Currently, the invoices are stored as files and processed by a desktop application. The application has several performance and security issues. The application is being migrated to a SQL Server-based solution. A schema named InvoiceSchema has been created for the invoices xml.

The data in the invoices is sometimes incomplete. The incomplete data must be stored and processed as-is. Users cannot filter the data provided through views.

You are designing a SQL Server database named DB1 that will be used to receive, process, and securely store the invoice data. A third-party Microsoft .NET Framework component will be purchased to perform tax calculations. The third-party tax component will be provided as a DLL file named Treytax.dll and a source code file named Amortize.cs. The component will expose a class named TreyResearch and a method named Amortize(). The files are located in c:\temp\.

The following graphic shows the planned tables:



You have a sequence named Accounting.InvoiceID_Seq.

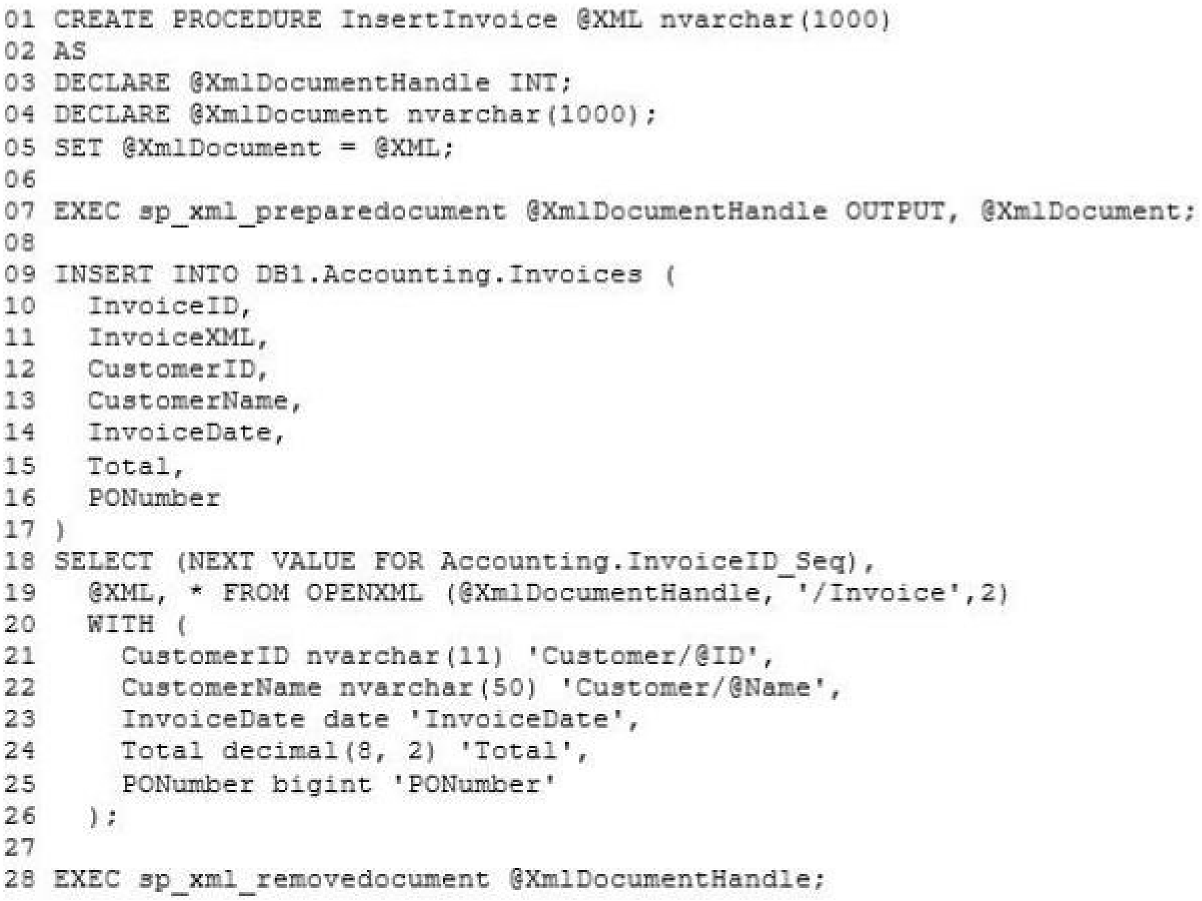
You plan to create two certificates named CERT1 and CERT2. You will create CERT1 in master. You will create CERT2 in DB1.

You have a legacy application that requires the ability to generate dynamic T-SQL statements against DB1. A sample of the queries generated by the legacy application appears in Legacy.sql.

Application Requirements

The planned database has the following requirements:
All tables must be disk-based
All stored procedures must be signed.
The original XML invoices must be stored in the database.
An XML schema must be used to validate the invoice data.
Dynamic T-SQL statements must be converted to stored procedures.
Access to the .NET Framework tax components must be available to T-SQL objects.
Columns must be defined by using data types that minimize the amount of space used by each table.
Invoices stored in the InvoiceStatus table must refer to an invoice by the same identifier used by the Invoice table.
To protect against the theft of backup disks, invoice data must be protected by using the highest level of encryption.
The solution must provide a table-valued function that provides users with the ability to filter invoices by customer.
Indexes must be optimized periodically based on their fragmentation by using the minimum amount of administrative effort.

Usp_InsertInvoices.sql



```
CREATE PROCEDURE InsertInvoice @XML nvarchar(1000)
AS
DECLARE @XmlDocumentHandle INT;
DECLARE @XmlDocument nvarchar(1000);
SET @XmlDocument = @XML;

EXEC sp_xml_preparedocument @XmlDocumentHandle OUTPUT, @XmlDocument;

INSERT INTO DB1.Accounting.Invoices (
  InvoiceID,
  InvoiceXML,
  CustomerID,
  CustomerName,
  InvoiceDate,
  Total,
  PONumber
)
SELECT (NEXT VALUE FOR Accounting.InvoiceID_Seq),
  @XML, * FROM OPENXML (@XmlDocumentHandle, '/Invoice',2)
  WITH (
    CustomerID nvarchar(11) 'Customer/@ID',
    CustomerName nvarchar(50) 'Customer/@Name',
    InvoiceDate date 'InvoiceDate',
    Total decimal(8, 2) 'Total',
    PONumber bigint 'PONumber'
  );

EXEC sp_xml_removedocument @XmlDocumentHandle;
```

Invoices.xml
All customer IDs are 11 digits. The first three digits of a customer ID represent the customer's country. The remaining eight digits are the customer's account number.

The following is a sample of a customer invoice in XML format:

```
<?xml version="1.0"?>
<Invoice InvoiceDate="2012-02-20">
  <Customer ID="00156590099" Name="Litware" />
  <Total>125</Total>
  <PONumber>1666</PONumber>
</Invoice>
```

InvoicesByCustomer.sql

```
(SELECT CustomerID,
  CustomerName,
  InvoiceID,
  InvoiceDate,
  Total,
  PONumber
  FROM Accounting.Invoices
  WHERE CustomerID=@CustID);
```

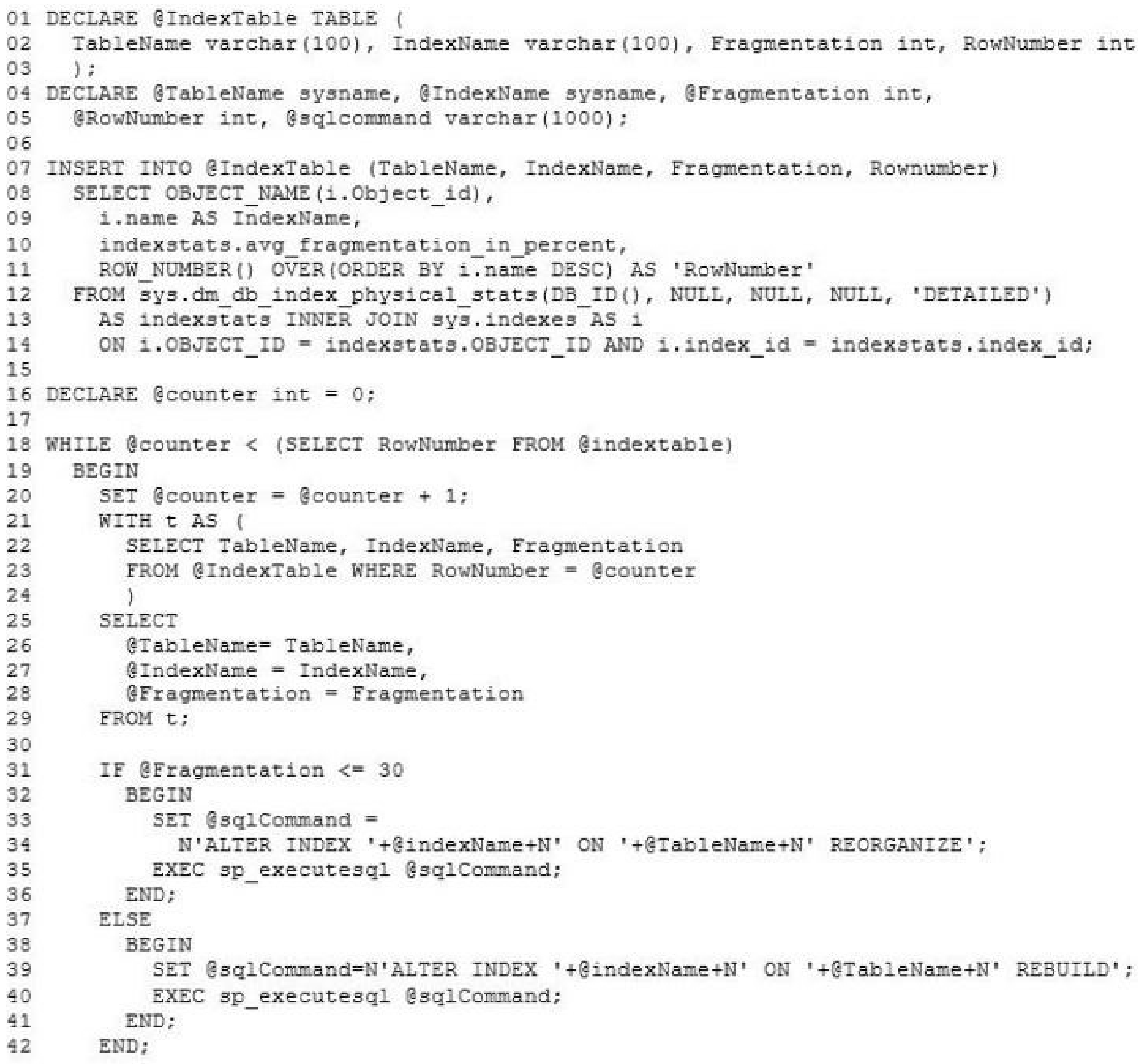
Legacy.sql

```
DECLARE @sqlstring AS nvarchar(1000);
DECLARE @CustomerID AS varchar(11), @Total AS decimal(8,2);

SET @sqlstring=N'SELECT CustomerID, InvoiceID, Total
  FROM Accounting.Invoices
  WHERE CustomerID=@CustomerID AND Total > @Total;';

EXEC sys.sp_executesql
  @statement=@sqlstring,
  @params=N'@CustomerID AS varchar(11), @Total AS decimal(8,2)',
  @CustomerID=999, @Total=500;
```

CountryFromID.sql

```
CREATE FUNCTION CountryFromID (@CustomerID varchar(11)) RETURNS varchar(20)
AS
BEGIN
  DECLARE @Country varchar(20);
  SET @CustomerID = LEFT(@CustomerID,3);
  SELECT @Country = CASE @CustomerID
    WHEN '001'
      THEN 'United States'
    WHEN '002'
      THEN 'Spain'
    WHEN '003'
      THEN 'Japan'
    WHEN '004'
      THEN 'China'
    WHEN '005'
      THEN 'Brazil'
    ELSE 'Other'
  END;
  RETURN @CustomerID;
END;
```

IndexManagement.sql

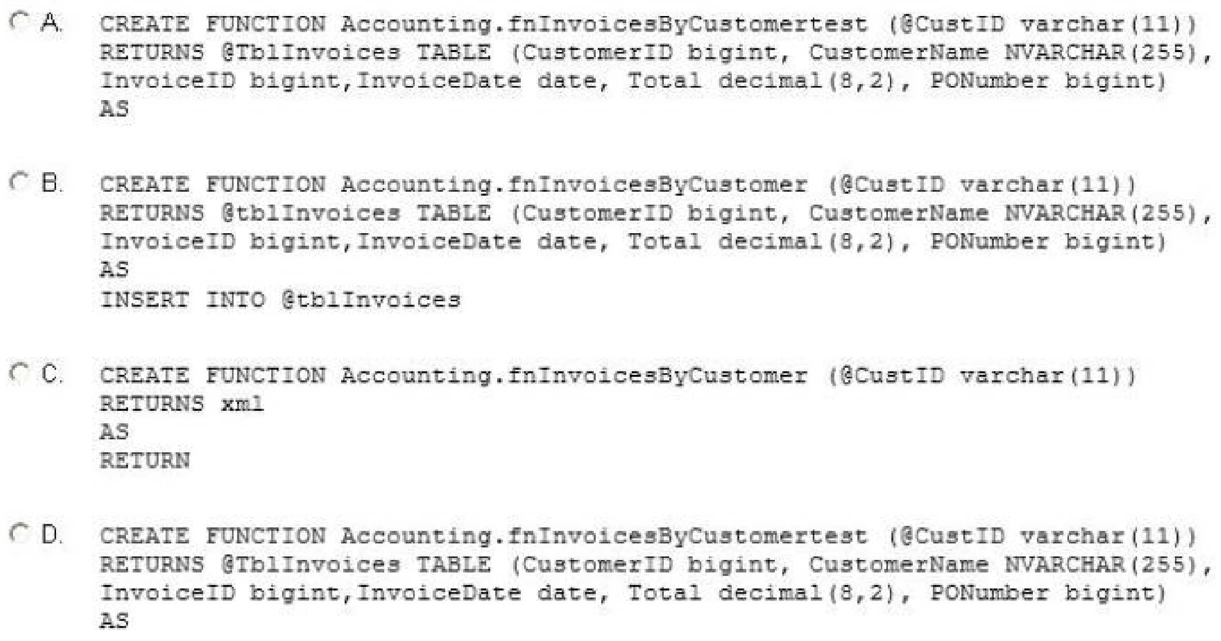


```
DECLARE @IndexTable TABLE (
  TableName varchar(100), IndexName varchar(100), Fragmentation int, RowNumber int
  );
DECLARE @TableName sysname, @IndexName sysname, @Fragmentation int,
  @RowNumber int, @sqlcommand varchar(1000);

INSERT INTO @IndexTable (TableName, IndexName, Fragmentation, Rownumber)
  SELECT OBJECT_NAME(i.Object_id),
    i.name AS IndexName,
    indexstats.avg_fragmentation_in_percent,
    ROW_NUMBER() OVER(ORDER BY i.name DESC) AS 'RowNumber'
  FROM sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, 'DETAILED')
    AS indexstats INNER JOIN sys.indexes AS i
    ON i.OBJECT_ID = indexstats.OBJECT_ID AND i.index_id = indexstats.index_id;

DECLARE @counter int = 0;

WHILE @counter < (SELECT RowNumber FROM @indextable)
  BEGIN
    SET @counter = @counter + 1;
    WITH t AS (
      SELECT TableName, IndexName, Fragmentation
      FROM @IndexTable WHERE RowNumber = @counter
      )
    SELECT
      @TableName= TableName,
      @IndexName = IndexName,
      @Fragmentation = Fragmentation
    FROM t;

    IF @Fragmentation <= 30
      BEGIN
        SET @sqlCommand =
          N'ALTER INDEX '+@indexName+N' ON '+@TableName+N' REORGANIZE';
        EXEC sp_executesql @sqlCommand;
      END;
    ELSE
      BEGIN
        SET @sqlCommand=N'ALTER INDEX '+@indexName+N' ON '+@TableName+N' REBUILD';
        EXEC sp_executesql @sqlCommand;
      END;
    END;
```

Question 1

You need to create a function that filters invoices by CustomerID. The SELECT statement for the function is contained in InvoicesByCustomer.sql.

Which code segment should you use to complete the function?



A.
```
CREATE FUNCTION Accounting.fnInvoicesByCustomertest (@CustID varchar(11))
RETURNS @TblInvoices TABLE (CustomerID bigint, CustomerName NVARCHAR(255),
InvoiceID bigint,InvoiceDate date, Total decimal(8,2), PONumber bigint)
AS
```

B.
```
CREATE FUNCTION Accounting.fnInvoicesByCustomer (@CustID varchar(11))
RETURNS @tblInvoices TABLE (CustomerID bigint, CustomerName NVARCHAR(255),
InvoiceID bigint,InvoiceDate date, Total decimal(8,2), PONumber bigint)
AS
INSERT INTO @tblInvoices
```

C.
```
CREATE FUNCTION Accounting.fnInvoicesByCustomer (@CustID varchar(11))
RETURNS xml
AS
RETURN
```

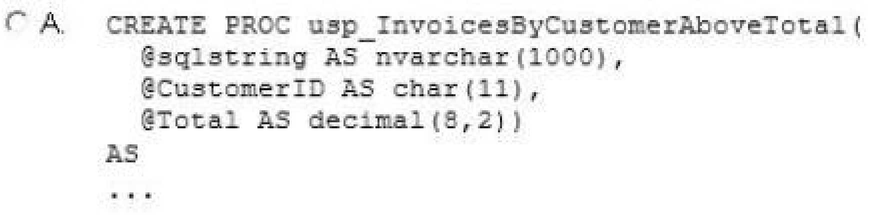
D.
```
CREATE FUNCTION Accounting.fnInvoicesByCustomertest (@CustID varchar(11))
RETURNS @TblInvoices TABLE (CustomerID bigint, CustomerName NVARCHAR(255),
InvoiceID bigint,InvoiceDate date, Total decimal(8,2), PONumber bigint)
AS
```

- ☐ Option A
- ☐ Option B
- ☐ Option C
- ☐ Option D

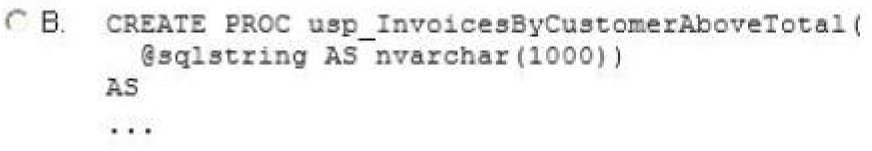
Question 2

You need to convert the functionality of Legacy.sql to use a stored procedure.

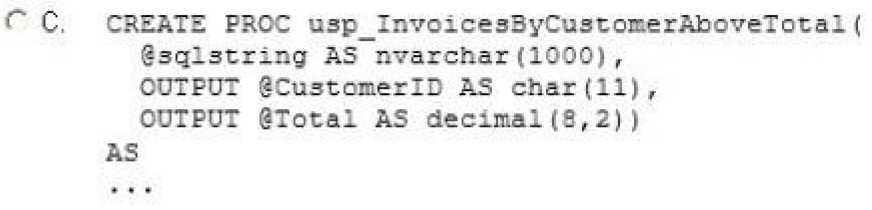
Which code segment should the stored procedure contain?



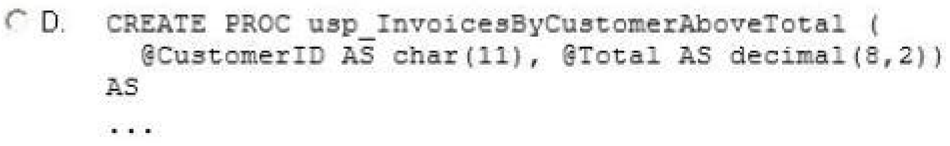
A.
```
CREATE PROC usp_InvoicesByCustomerAboveTotal(
  @sqlstring AS nvarchar(1000),
  @CustomerID AS char(11),
  @Total AS decimal(8,2))
AS
...
```



B.
```
CREATE PROC usp_InvoicesByCustomerAboveTotal(
  @sqlstring AS nvarchar(1000))
AS
...
```



C.
```
CREATE PROC usp_InvoicesByCustomerAboveTotal(
  @sqlstring AS nvarchar(1000),
  OUTPUT @CustomerID AS char(11),
  OUTPUT @Total AS decimal(8,2))
AS
...
```



D.
```
CREATE PROC usp_InvoicesByCustomerAboveTotal (
  @CustomerID AS char(11), @Total AS decimal(8,2))
AS
...
```

- ☐ Option A
- ☐ Option B
- ☐ Option C
- ☐ Option D

Question 3

You need to modify the function in CountryFromID.sql to ensure that the country name is returned instead of the country ID.

Which line of code should you modify in CountryFromID.sql?

- ☐ 04
- ☐ 05
- ☐ 06
- ☐ 19

Question 4

You execute IndexManagement.sql and you receive the following error message:

"Msg 512, Level 16, State 1, Line 12
Subquery returned more than 1 value. This is not permitted when the subquery follows =,! =, <, <= ,>, > = or when the subquery is used as an expression."

You need to ensure that IndexManagement.sql executes properly.

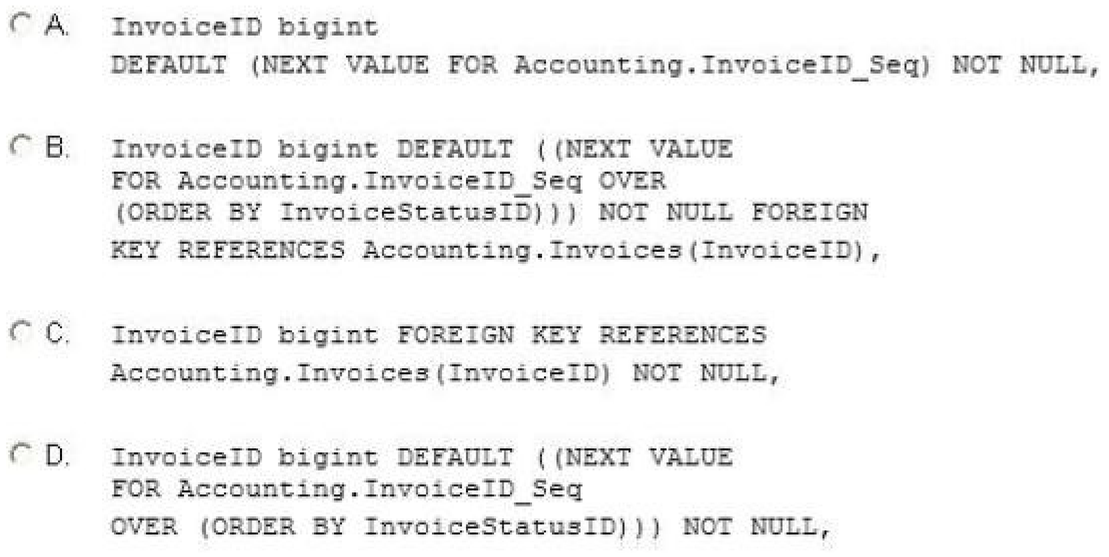
Which WHILE statement should you use at line 18?

- ☐ WHILE SUM(@RowNumber) < (SELECT @counter FROM @indextable)
- ☐ WHILE @counter < (SELECT COUNT(RowNumber) FROM @indextable)
- ☐ WHILE COUNT(@RowNumber) < (SELECT @counter FROM @indextable)
- ☐ WHILE @counter < (SELECT SUM(RowNumber) FROM @indextabie)

Question 5

You need to create the InvoiceStatus table in DB1.

How should you define the InvoiceID column in the CREATE TABLE statement?



A.
```
InvoiceID bigint
DEFAULT (NEXT VALUE FOR Accounting.InvoiceID_Seq) NOT NULL,
```

B.
```
InvoiceID bigint DEFAULT ((NEXT VALUE
FOR Accounting.InvoiceID_Seq OVER
(ORDER BY InvoiceStatusID))) NOT NULL FOREIGN
KEY REFERENCES Accounting.Invoices(InvoiceID),
```

C.
```
InvoiceID bigint FOREIGN KEY REFERENCES
Accounting.Invoices(InvoiceID) NOT NULL,
```

D.
```
InvoiceID bigint DEFAULT ((NEXT VALUE
FOR Accounting.InvoiceID_Seq
OVER (ORDER BY InvoiceStatusID))) NOT NULL,
```

- ☐ Option A
- ☐ Option B
- ☐ Option C
- ☐ Option D

Topic 2, Scenario 2

Case Study (5 questions)

Scenario 2

Application Information

You have two servers named SQL1 and SQL2 that have SQL Server 2012 installed.

You have an application that is used to schedule and manage conferences.

Users report that the application has many errors and is very slow.

You are updating the application to resolve the issues.

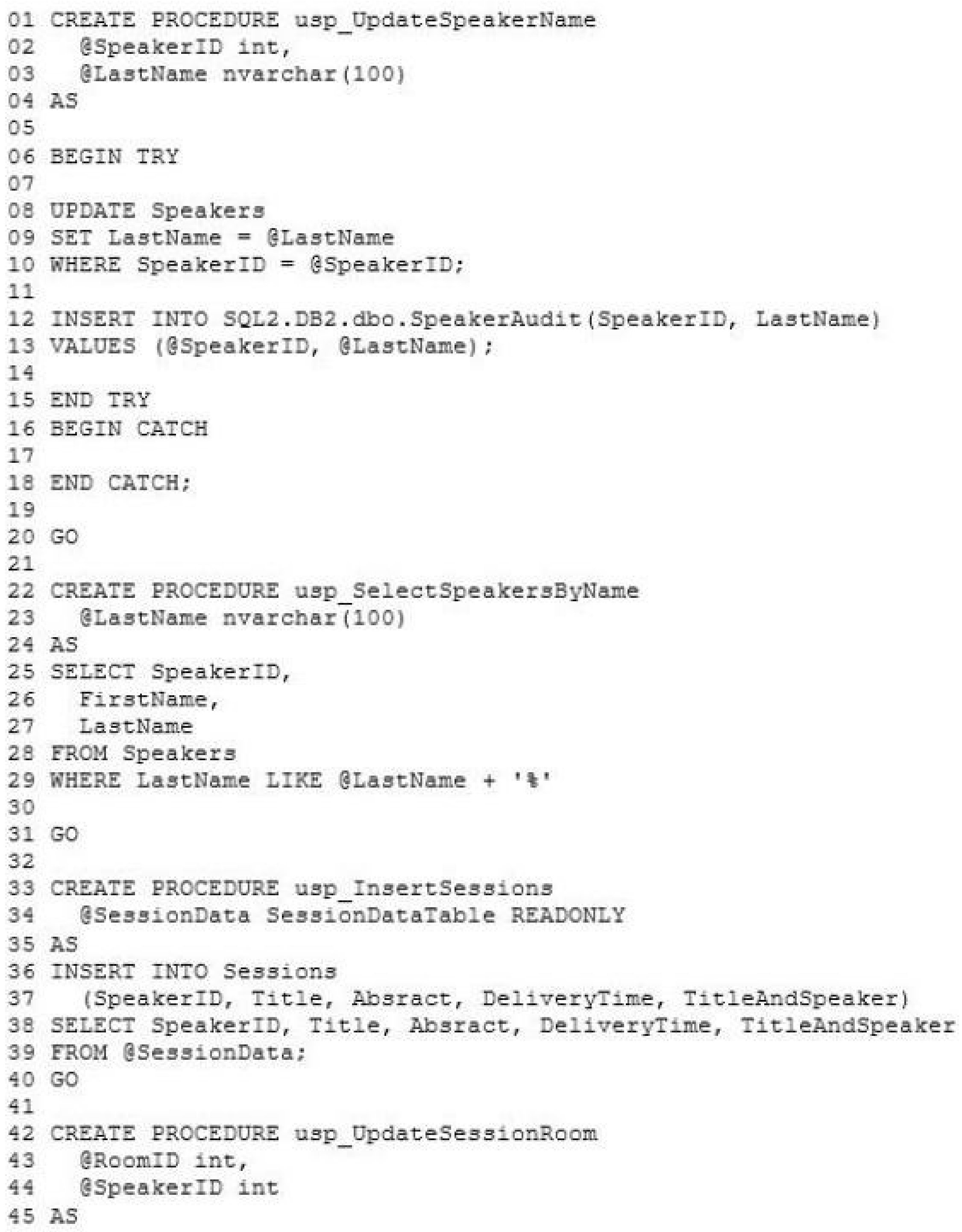
You plan to create a new database on SQL1 to support the application. A junior database administrator has created all the scripts that will be used to create the database. The script that you plan to use to create the tables for the new database is shown in Tables.sql. The script that you plan to use to create the stored procedures for the new database is shown in StoredProcedures.sql. The script that you plan to use to create the indexes for the new database is shown in Indexes.sql. (Line numbers are included for reference only.)

A database named DB2 resides on SQL2. DB2 has a table named SpeakerAudit that will audit changes to a table named Speakers.
A stored procedure named usp_UpdateSpeakersName will be executed only by other stored procedures. The stored procedures executing usp_UpdateSpeakersName will always handle transactions.

A stored procedure named usp_SelectSpeakersByName will be used to retrieve the names of speakers. Usp_SelectSpeakersByName can read uncommitted data.

A stored procedure named usp_GetFutureSessions will be used to retrieve sessions that will occur in the future.

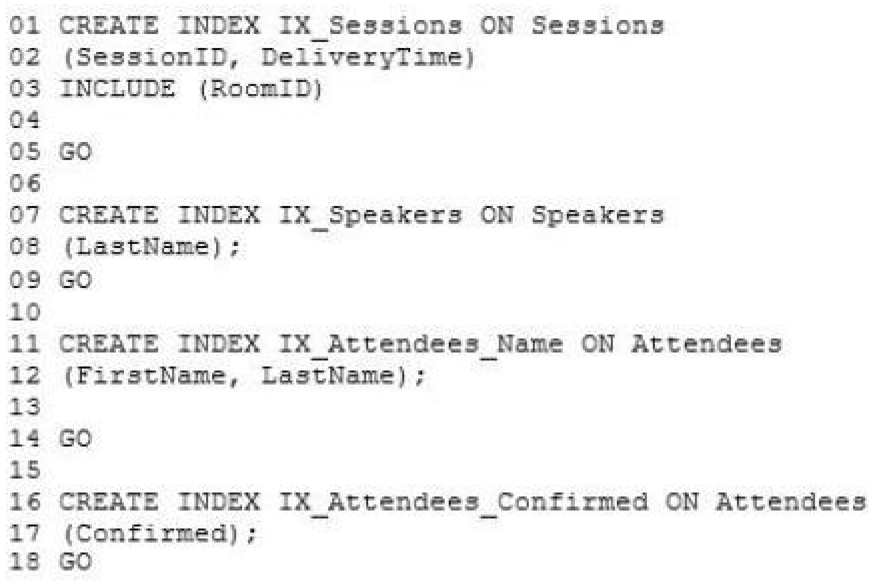
Procedures.sql



```
CREATE PROCEDURE usp_UpdateSpeakerName
  @SpeakerID int,
  @LastName nvarchar(100)
AS

BEGIN TRY

UPDATE Speakers
SET LastName = @LastName
WHERE SpeakerID = @SpeakerID;

INSERT INTO SQL2.DB2.dbo.SpeakerAudit(SpeakerID, LastName)
VALUES (@SpeakerID, @LastName);

END TRY
BEGIN CATCH

END CATCH;

GO

CREATE PROCEDURE usp_SelectSpeakersByName
  @LastName nvarchar(100)
AS
SELECT SpeakerID,
  FirstName,
  LastName
FROM Speakers
WHERE LastName LIKE @LastName + '%'

GO

CREATE PROCEDURE usp_InsertSessions
  @SessionData SessionDataTable READONLY
AS
INSERT INTO Sessions
  (SpeakerID, Title, Absract, DeliveryTime, TitleAndSpeaker)
SELECT SpeakerID, Title, Absract, DeliveryTime, TitleAndSpeaker
FROM @SessionData;
GO

CREATE PROCEDURE usp_UpdateSessionRoom
  @RoomID int,
  @SpeakerID int
AS
```

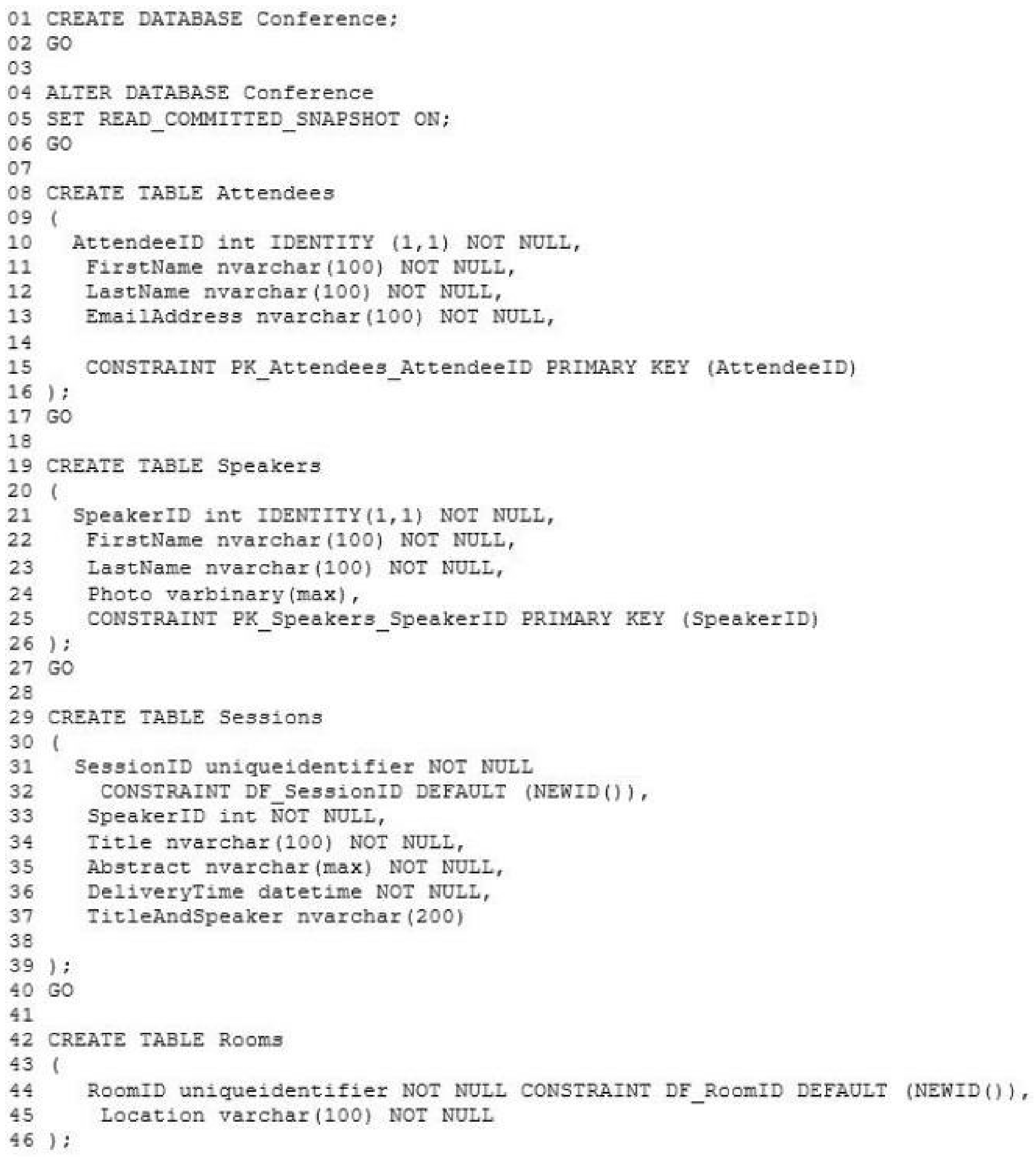
```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT
BEGIN TRANSACTION;

SELECT SessionID,
  Title
FROM Sessions
WHERE SpeakerID = @SpeakerID;

UPDATE Sessions
SET RoomID = @RoomID
WHERE SpeakerID = @SpeakerID;

COMMIT TRANSACTION;

CREATE PROCEDURE usp_AttendeesReport
  @LastName varchar(100)
AS
SELECT FirstName + ' ' + LastName AS FullName
FROM Attendees
WHERE LastName = @LastName;
GO

CREATE PROCEDURE usp_GetFutureSessions
AS
SELECT SpeakerID,
  RoomID,
  DeliveryTime
FROM Sessions

GO

CREATE PROCEDURE usp_TestSpeakers
AS
EXECUTE usp_SelectSpeakersByName 'a';
EXECUTE usp_SelectSpeakersByName 'an';
EXECUTE usp_SelectSpeakersByName 'and';
EXECUTE usp_SelectSpeakersByName 'ander';
EXECUTE usp_SelectSpeakersByName 'anderson';
EXECUTE usp_SelectSpeakersByName 'b';
EXECUTE usp_SelectSpeakersByName 'bi';
...
EXECUTE usp_SelectSpeakersByName 'zzz';
GO
```

Indexes.sql



```sql
CREATE INDEX IX_Sessions ON Sessions
(SessionID, DeliveryTime)
INCLUDE (RoomID)

GO

CREATE INDEX IX_Speakers ON Speakers
(LastName);
GO

CREATE INDEX IX_Attendees_Name ON Attendees
(FirstName, LastName);

GO

CREATE INDEX IX_Attendees_Confirmed ON Attendees
(Confirmed);
GO
```

Tables.sql



```
CREATE DATABASE Conference;
GO

ALTER DATABASE Conference
SET READ_COMMITTED_SNAPSHOT ON;
GO

CREATE TABLE Attendees
(
  AttendeeID int IDENTITY (1,1) NOT NULL,
   FirstName nvarchar(100) NOT NULL,
   LastName nvarchar(100) NOT NULL,
   EmailAddress nvarchar(100) NOT NULL,

   CONSTRAINT PK_Attendees_AttendeeID PRIMARY KEY (AttendeeID)
);
GO

CREATE TABLE Speakers
(
  SpeakerID int IDENTITY(1,1) NOT NULL,
   FirstName nvarchar(100) NOT NULL,
   LastName nvarchar(100) NOT NULL,
   Photo varbinary(max),
   CONSTRAINT PK_Speakers_SpeakerID PRIMARY KEY (SpeakerID)
);
GO

CREATE TABLE Sessions
(
  SessionID uniqueidentifier NOT NULL
     CONSTRAINT DF_SessionID DEFAULT (NEWID()),
   SpeakerID int NOT NULL,
   Title nvarchar(100) NOT NULL,
   Abstract nvarchar(max) NOT NULL,
   DeliveryTime datetime NOT NULL,
   TitleAndSpeaker nvarchar(200)

);
GO

CREATE TABLE Rooms
(
   RoomID uniqueidentifier NOT NULL CONSTRAINT DF_RoomID DEFAULT (NEWID()),
    Location varchar(100) NOT NULL
);
```

Question 10

You need to add a new column named Confirmed to the Attendees table.

The solution must meet the following requirements:
Have a default value of false.
Minimize the amount of disk space used.

Which code block should you use?

- ☐ ALTER TABLE Attendees

- ☐ ADD Confirmed bit DEFAULT 0;
- ☐ ALTER TABLE Attendees
- ☐ ADD Confirmed char(l) DEFAULT '1';
- ☐ ALTER TABLE Attendees
- ☐ ADD Confirmed bit DEFAULT 1;
- ☐ ALTER TABLE Attendees
- ☐ ADD Confirmed char(l) DEFAULT `1';

Question 11

Developers report that usp_UpdateSessionRoom periodically returns error 3960.

You need to prevent the error from occurring. The solution must ensure that the stored procedure returns the original values to all of the updated rows.

What should you configure in Procedures.sql?

- ☐ Replace line 46 with the following code: SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
- ☐ Replace line 46 with the following code: SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
- ☐ Move the SELECT statement at line 49 to line 57.
- ☐ Move the SET statement at line 46 to line 53.

Question 12

You need to create the object used by the parameter of usp_InsertSessions.

Which statement should you use?

- ☐ CREATE XML SCHEMA COLLECTION SessionDataTable
- ☐ CREATE TYPE SessionDataTable AS Table
- ☐ CREATE SCHEMA SessionDataTable
- ☐ CREATE TABLE SessionDataTable

Question 13

You are evaluating the index design for the database.

You have the following requirements:
minimize the amount of time it takes for usp_AttendeesReport to run
minimize the amount of database fragmentation.

You need to recommend a change to Indexes.sql

Which line of code should you use to replace line 12 of Indexes.sql?

- ☐ (LastName);
- ☐ (FirstName) INCLUDE (LastName);
- ☐ (LastName, FirstName);
- ☐ (LastName) INCLUDE (FirstName);

Question 14

You are evaluating the table design.

You need to recommend a change to Tables.sql that reduces the amount of time it takes for usp_AttendeesReport to execute.

What should you add at line 14 of Tables.sql?

- ☐ FullName nvarchar(100) NOT NULL CONSTRAINT DF_FullName DEFAULT (dbo.CreateFullName (FirstName, LastName)),
- ☐ FullName AS (FirstName +` '+ LastName),
- ☐ FullName nvarchar(100) NOT NULL DEFAULT (dbo.CreateFullName (FirstName, LastName)).
- ☐ FullName AS (FirstName +` '+ LastName) PERSISTED,

Topic 3, Scenario 3

Case Study (5 questions)

Scenario 3

Application Information

You have two servers named SQL1 and SQL2. SQL1 has SQL Server 2012 Enterprise installed.
SQL2 has SQL Server 2008 Standard installed.

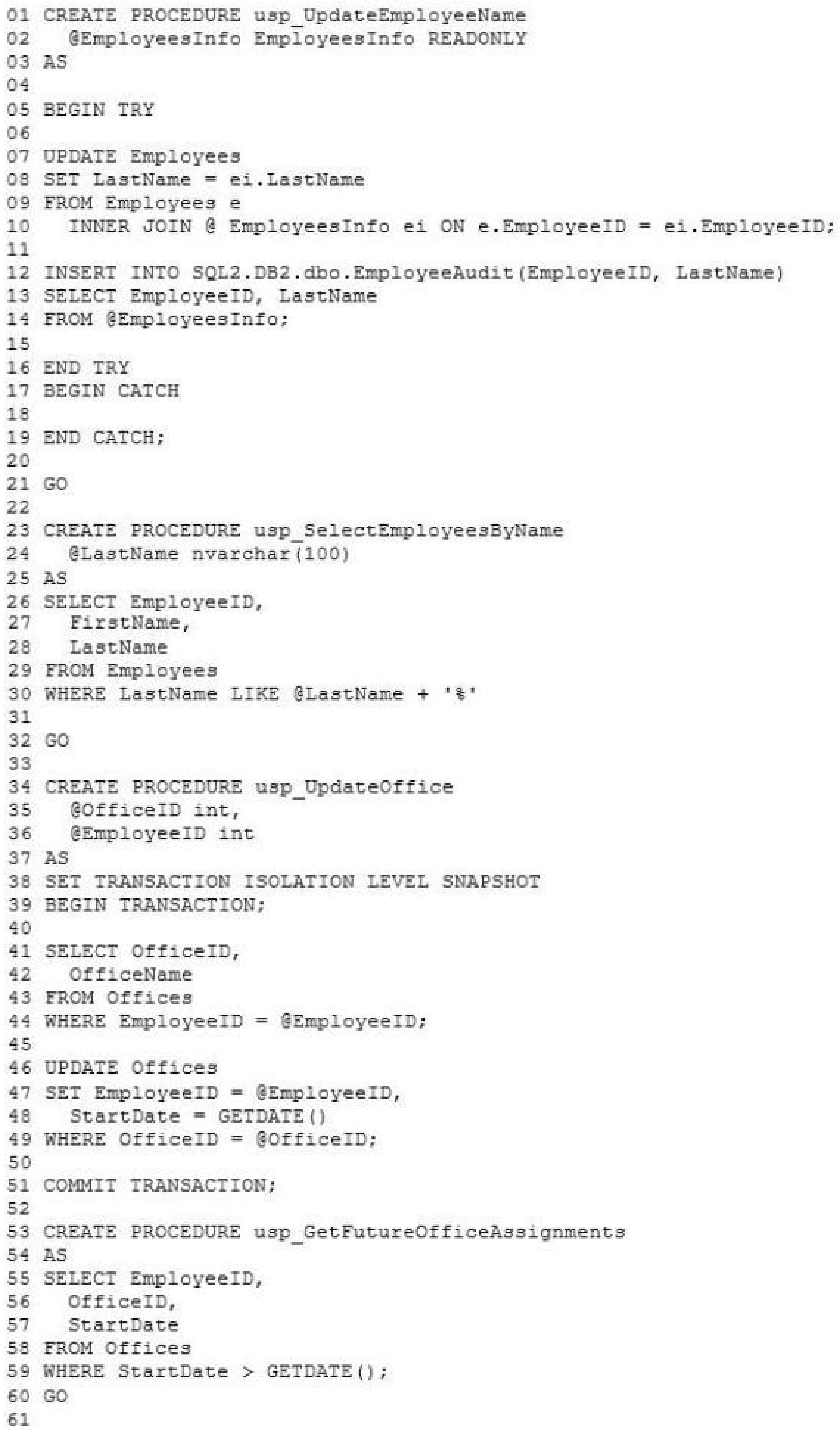
You have an application that is used to manage employees and office space. Users report that the application has many errors and is very slow.

You are updating the application to resolve the issues. You plan to create a new database on SQL1 to support the application. The script that you plan to use to create the tables for the new database is shown in Tables.sql. The script that you plan to use to create the stored procedures for the new database is shown in StoredProcedures.sql. The script that you plan to use to create the indexes for the new database is shown in Indexes.sql. A database named DB2 resides on SQL2. DB2 has a table named EmployeeAudit that will audit changes to a table named Employees.

A stored procedure named usp_UpdateEmployeeName will be executed only by other stored procedures. The stored procedures executing usp_UpdateEmp!oyeeName will always handle transactions.

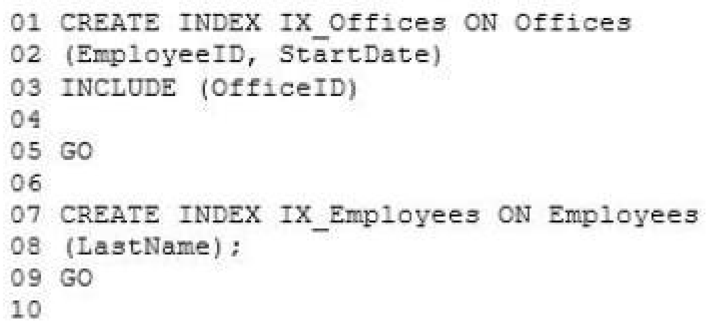
A stored procedure named usp_SelectEmployeesByName will be used to retrieve the names of employees. Usp_SelectEmployeesByName can read uncommitted data. A stored procedure named usp_GetFutureOfficeAssignments will be used to retrieve office assignments that will occur in the future.

StoredProcedures.sql



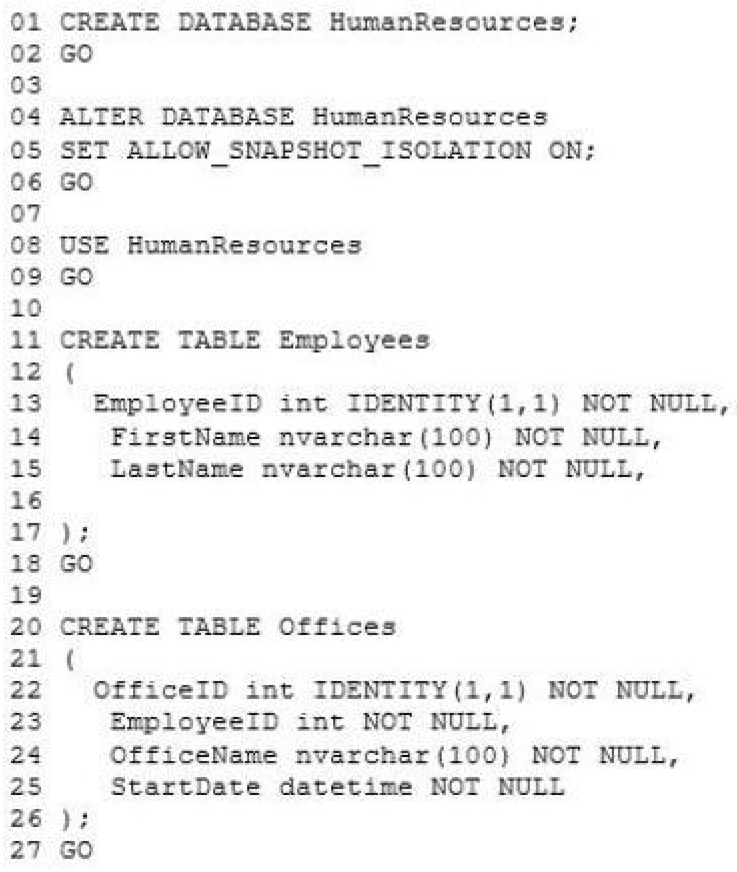
```
CREATE PROCEDURE usp_UpdateEmployeeName
  @EmployeesInfo EmployeesInfo READONLY
AS

BEGIN TRY

UPDATE Employees
SET LastName = ei.LastName
FROM Employees e
  INNER JOIN @ EmployeesInfo ei ON e.EmployeeID = ei.EmployeeID;

INSERT INTO SQL2.DB2.dbo.EmployeeAudit(EmployeeID, LastName)
SELECT EmployeeID, LastName
FROM @EmployeesInfo;

END TRY
BEGIN CATCH

END CATCH;

GO

CREATE PROCEDURE usp_SelectEmployeesByName
  @LastName nvarchar(100)
AS
SELECT EmployeeID,
  FirstName,
  LastName
FROM Employees
WHERE LastName LIKE @LastName + '%'

GO

CREATE PROCEDURE usp_UpdateOffice
  @OfficeID int,
  @EmployeeID int
AS
SET TRANSACTION ISOLATION LEVEL SNAPSHOT
BEGIN TRANSACTION;

SELECT OfficeID,
  OfficeName
FROM Offices
WHERE EmployeeID = @EmployeeID;

UPDATE Offices
SET EmployeeID = @EmployeeID,
  StartDate = GETDATE()
WHERE OfficeID = @OfficeID;

COMMIT TRANSACTION;

CREATE PROCEDURE usp_GetFutureOfficeAssignments
AS
SELECT EmployeeID,
  OfficeID,
  StartDate
FROM Offices
WHERE StartDate > GETDATE();
GO

```

Indexes.sql



```
CREATE INDEX IX_Offices ON Offices
(EmployeeID, StartDate)
INCLUDE (OfficeID)

GO

CREATE INDEX IX_Employees ON Employees
(LastName);
GO

```

Tables.sql

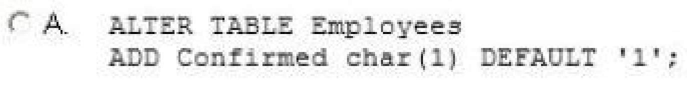


```
CREATE DATABASE HumanResources;
GO

ALTER DATABASE HumanResources
SET ALLOW_SNAPSHOT_ISOLATION ON;
GO

USE HumanResources
GO

CREATE TABLE Employees
(
  EmployeeID int IDENTITY(1,1) NOT NULL,
   FirstName nvarchar(100) NOT NULL,
   LastName nvarchar(100) NOT NULL,

);
GO

CREATE TABLE Offices
(
  OfficeID int IDENTITY(1,1) NOT NULL,
   EmployeeID int NOT NULL,
   OfficeName nvarchar(100) NOT NULL,
   StartDate datetime NOT NULL
);
GO
```

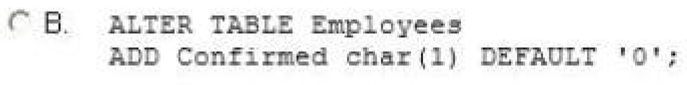
Question 22

You need to add a new column named Confirmed to the Employees table. The solution must meet the following requirements:
Have a default value of TRUE.
Minimize the amount of disk space used.

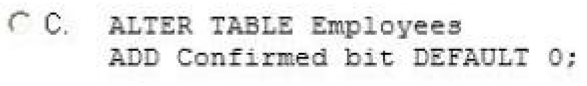
Which code segment should you use?



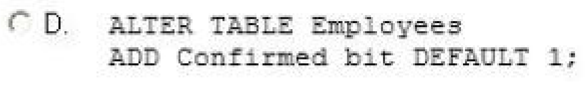
```
◯ A.  ALTER TABLE Employees
      ADD Confirmed char(1) DEFAULT '1';
```



```
◯ B.  ALTER TABLE Employees
      ADD Confirmed char(1) DEFAULT '0';
```



```
◯ C.  ALTER TABLE Employees
      ADD Confirmed bit DEFAULT 0;
```



```
◯ D.  ALTER TABLE Employees
      ADD Confirmed bit DEFAULT 1;
```

- ☐ Option A
- ☐ Option B
- ☐ Option C
- ☐ Option D

Question 23

You need to recommend a solution to ensure that SQL1 supports the auditing requirements of usp_UpdateEmployeeName.

What should you include in the recommendation?

- ☐ Change data capture
- ☐ Change tracking
- ☐ Transactional replication
- ☐ The Distributed Transaction Coordinator (DTC)

Question 24

You execute usp_SelectEmployeesByName multiple times, passing strings of varying lengths to @LastName. You discover that usp_SelectEmployeesByName uses inefficient execution plans.

You need to update usp_SelectEmployeesByName to ensure that the most efficient execution plan is used.
What should you add at line 31 of StoredProcedures.sql?

- ☐ OPTION (ROBUST PLAN)
- ☐ OPTION (OPTIMIZE FOR UNKNOWN)
- ☐ OPTION (KEEP PLAN)
- ☐ OPTION (KEEPFIXED PLAN)

Question 25

You need to create the object used by the parameter of usp_UpdateEmployeeName.

Which code segment should you use?

- ☐ CREATE XML SCHEMA COLLECTION EmployeesInfo
- ☐ CREATE TYPE EmployeesInfo AS Table
- ☐ CREATE SCHEMA EmployeesInfo
- ☐ CREATE TABLE EmployeesInfo

Question 26

You need to modify usp_SelectEmployeesByName to support server-side paging. The solution must minimize the amount of development effort required.

What should you add to usp_SelectEmployeesByName?

- ☐ A table variable
- ☐ The ROWNUMBER keyword
- ☐ An OFFSET-FETCH clause
- ☐ A recursive common table expression

Topic 4, Scenario 4

Case Study (5 questions)

Scenario 4

Application Information

You are a database administrator for a manufacturing company.

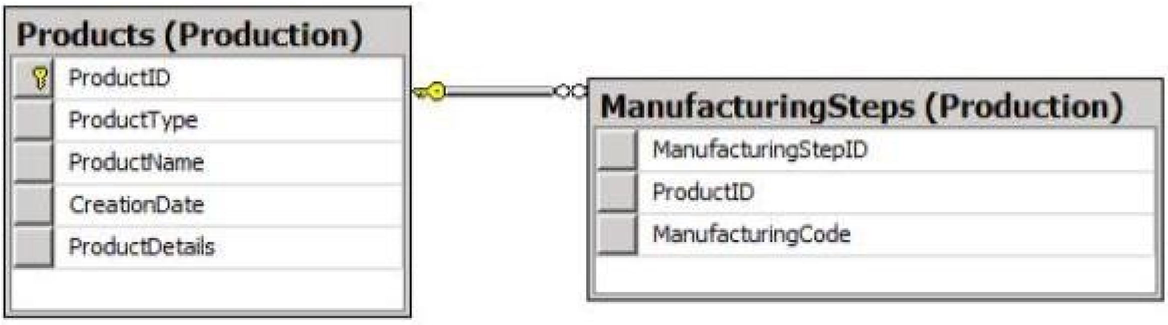
You have an application that stores product data. The data will be converted to technical diagrams for the manufacturing process.

The product details are stored in XML format. Each XML must contain only one product that has a root element named Product. A schema named Production.ProductSchema has been created for the products xml.

You develop a Microsoft .NET Framework assembly named ProcessProducts.dll that will be used to convert the XML files to diagrams. The diagrams will be stored in the database as images. ProcessProducts.dll contains one class named ProcessProduct that has a method name of Convert(). ProcessProducts.dll was created by using a source code file named ProcessProduct.cs.

All of the files are located in C:\Products\.
The application has several performance and security issues. You will create a new database named ProductsDB on a new server that has SQL Server 2014 installed. ProductsDB will support the application.

The following graphic shows the planned tables for ProductsDB:



You will also add a sequence named Production.ProductID_Seq.

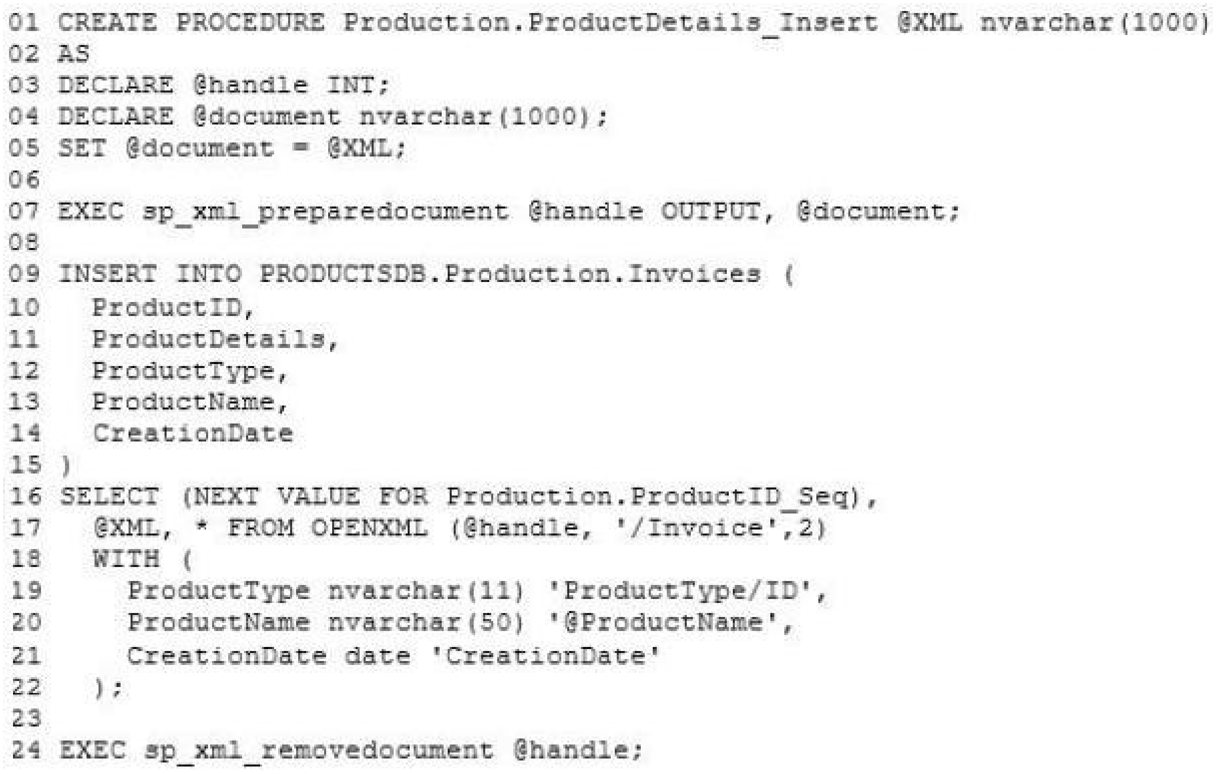
You plan to create two certificates named DBCert and ProductsCert. You will create ProductsCert in master. You will create DBCert in ProductsDB.

You have an application that executes dynamic T-SQL statements against ProductsDB. A sample of the queries generated by the application appears in Dynamic.sql.

Application Requirements

The planned database has the following requirements:
All tables must be disk-based
All stored procedures must be signed.
The amount of disk space must be minimized.
Administrative effort must be minimized at all times.
The original product details must be stored in the database.
An XML schema must be used to validate the product details.
The assembly must be accessible by using T-SQL commands.
A table-valued function will be created to search products by type.
Backups must be protected by using the highest level of encryption.
Dynamic T-SQL statements must be converted to stored procedures.
Indexes must be optimized periodically based on their fragmentation.
Manufacturing steps stored in the ManufacturingSteps table must refer to a product by the same identifier used by the Products table.

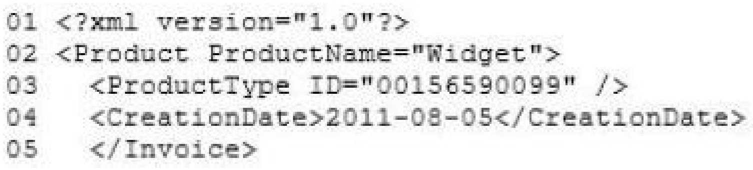
ProductDetails_Insert.sql



```
CREATE PROCEDURE Production.ProductDetails_Insert @XML nvarchar(1000)
AS
DECLARE @handle INT;
DECLARE @document nvarchar(1000);
SET @document = @XML;

EXEC sp_xml_preparedocument @handle OUTPUT, @document;

INSERT INTO PRODUCTSDB.Production.Invoices (
  ProductID,
  ProductDetails,
  ProductType,
  ProductName,
  CreationDate
)
SELECT (NEXT VALUE FOR Production.ProductID_Seq),
  @XML, * FROM OPENXML (@handle, '/Invoice',2)
  WITH (
    ProductType nvarchar(11) 'ProductType/ID',
    ProductName nvarchar(50) '@ProductName',
    CreationDate date 'CreationDate'
  );

EXEC sp_xml_removedocument @handle;
```

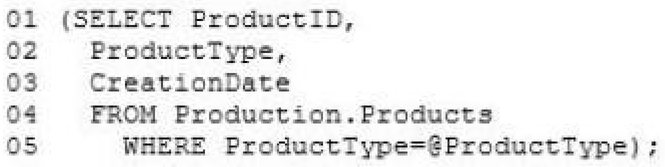
Product, xml
All product types are 11 digits. The first five digits of the product id reference the category of the product and the remaining six digits are the subcategory of the product.

The following is a sample customer invoice in XML format:



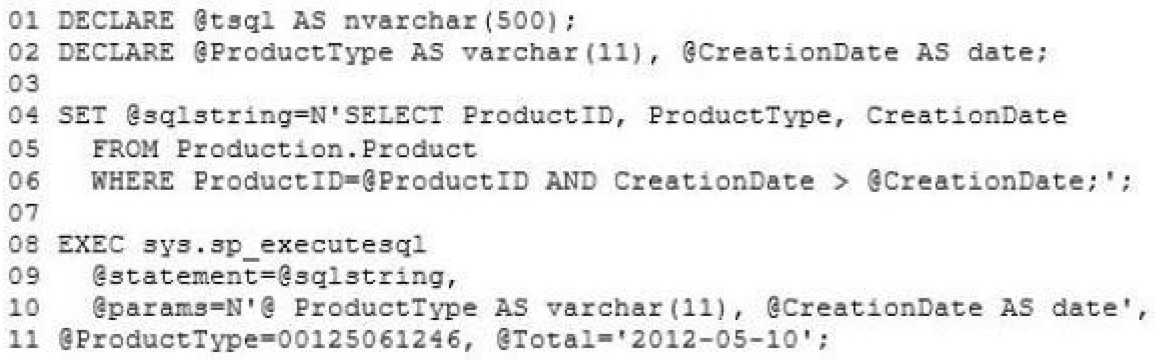
```
<?xml version="1.0"?>
<Product ProductName="Widget">
  <ProductType ID="00156590099" />
  <CreationDate>2011-08-05</CreationDate>
  </Invoice>
```

ProductsByProductType.sql



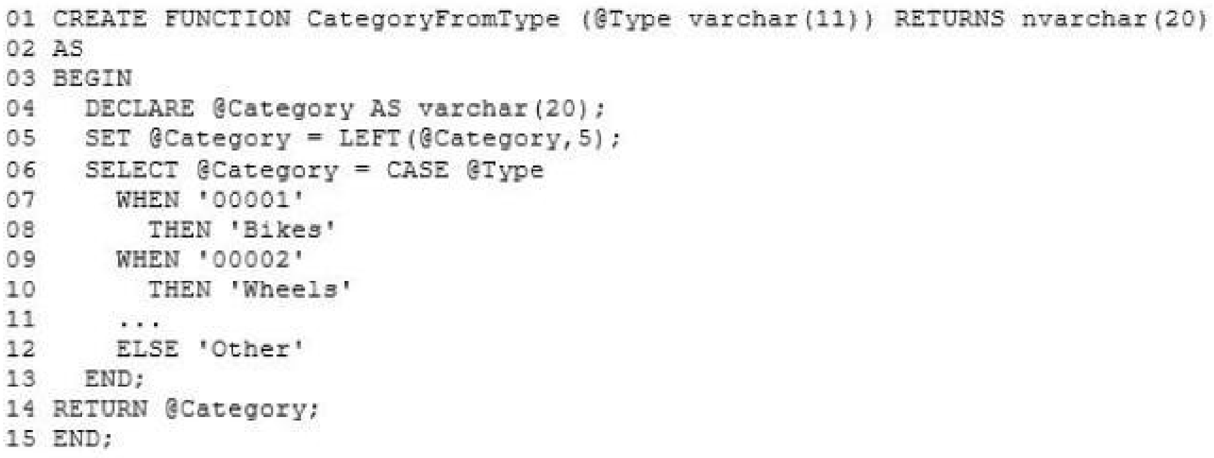
```
(SELECT ProductID,
  ProductType,
  CreationDate
  FROM Production.Products
    WHERE ProductType=@ProductType);
```

Dynamic.sql



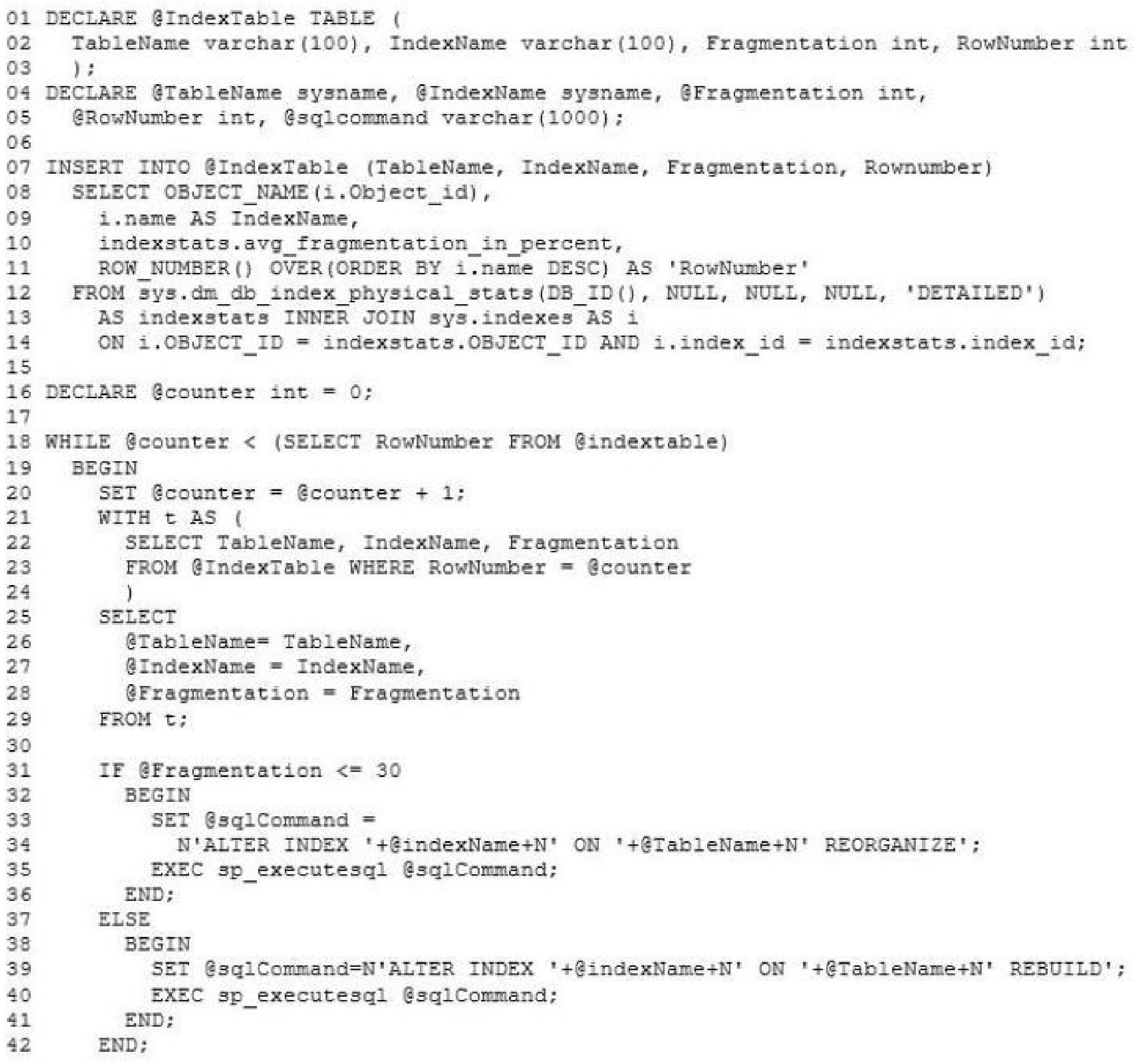
```
DECLARE @tsql AS nvarchar(500);
DECLARE @ProductType AS varchar(11), @CreationDate AS date;

SET @sqlstring=N'SELECT ProductID, ProductType, CreationDate
  FROM Production.Product
  WHERE ProductID=@ProductID AND CreationDate > @CreationDate;';

EXEC sys.sp_executesql
  @statement=@sqlstring,
  @params=N'@ ProductType AS varchar(11), @CreationDate AS date',
@ProductType=00125061246, @Total='2012-05-10';
```

Category FromType.sql



```
CREATE FUNCTION CategoryFromType (@Type varchar(11)) RETURNS nvarchar(20)
AS
BEGIN
  DECLARE @Category AS varchar(20);
  SET @Category = LEFT(@Category,5);
  SELECT @Category = CASE @Type
    WHEN '00001'
      THEN 'Bikes'
    WHEN '00002'
      THEN 'Wheels'
    ...
    ELSE 'Other'
  END;
RETURN @Category;
END;
```

IndexManagement.sql

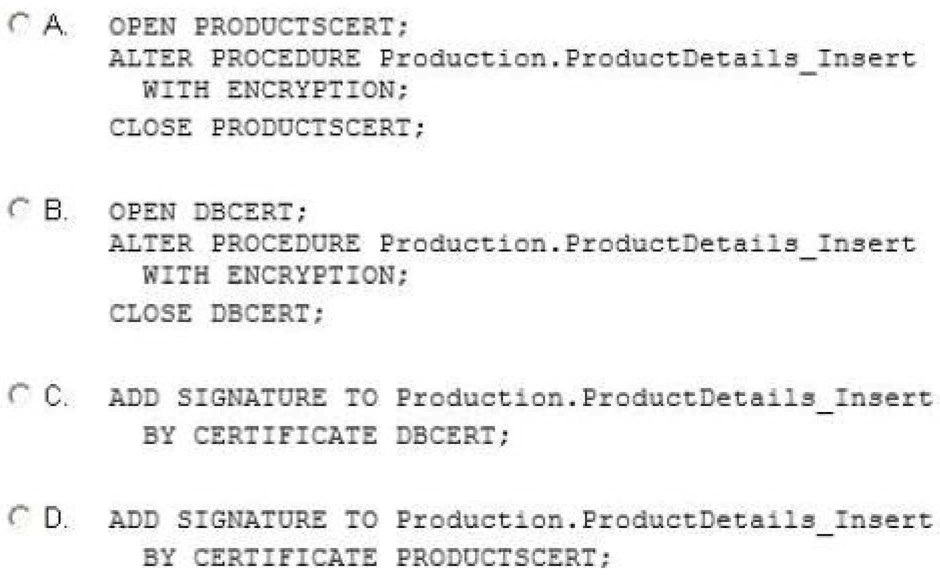


```
DECLARE @IndexTable TABLE (
  TableName varchar(100), IndexName varchar(100), Fragmentation int, RowNumber int
  );
DECLARE @TableName sysname, @IndexName sysname, @Fragmentation int,
  @RowNumber int, @sqlcommand varchar(1000);

INSERT INTO @IndexTable (TableName, IndexName, Fragmentation, Rownumber)
  SELECT OBJECT_NAME(i.Object_id),
    i.name AS IndexName,
    indexstats.avg_fragmentation_in_percent,
    ROW_NUMBER() OVER(ORDER BY i.name DESC) AS 'RowNumber'
  FROM sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, 'DETAILED')
    AS indexstats INNER JOIN sys.indexes AS i
    ON i.OBJECT_ID = indexstats.OBJECT_ID AND i.index_id = indexstats.index_id;

DECLARE @counter int = 0;

WHILE @counter < (SELECT RowNumber FROM @indextable)
  BEGIN
    SET @counter = @counter + 1;
    WITH t AS (
      SELECT TableName, IndexName, Fragmentation
      FROM @IndexTable WHERE RowNumber = @counter
      )
    SELECT
      @TableName= TableName,
      @IndexName = IndexName,
      @Fragmentation = Fragmentation
    FROM t;

    IF @Fragmentation <= 30
      BEGIN
        SET @sqlCommand =
          N'ALTER INDEX '+@indexName+N' ON '+@TableName+N' REORGANIZE';
        EXEC sp_executesql @sqlCommand;
      END;
    ELSE
      BEGIN
        SET @sqlCommand=N'ALTER INDEX '+@indexName+N' ON '+@TableName+N' REBUILD';
        EXEC sp_executesql @sqlCommand;
      END;
    END;
```

Question 31

You need to modify Production.ProductDetails_Insert to comply with the application requirements.

Which code segment should you execute?



A.
```
OPEN PRODUCTSCERT;
ALTER PROCEDURE Production.ProductDetails_Insert
  WITH ENCRYPTION;
CLOSE PRODUCTSCERT;
```

B.
```
OPEN DBCERT;
ALTER PROCEDURE Production.ProductDetails_Insert
  WITH ENCRYPTION;
CLOSE DBCERT;
```

C.
```
ADD SIGNATURE TO Production.ProductDetails_Insert
  BY CERTIFICATE DBCERT;
```

D.
```
ADD SIGNATURE TO Production.ProductDetails_Insert
  BY CERTIFICATE PRODUCTSCERT;
```

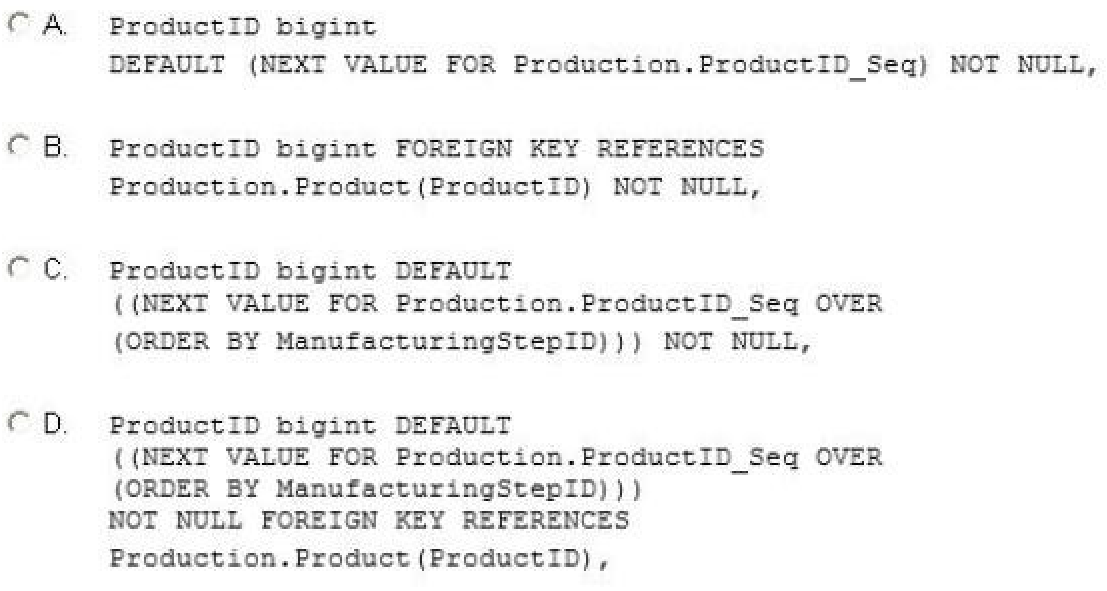
- ☐ Option A
- ☐ Option B
- ☐ Option C
- ☐ Option D

Question 32

You are planning the ManufacturingSteps table.

You need to define the ProductID column in the CREATE TABLE statement.

Which code segment should you use?



A.
```
ProductID bigint
DEFAULT (NEXT VALUE FOR Production.ProductID_Seq) NOT NULL,
```

B.
```
ProductID bigint FOREIGN KEY REFERENCES
Production.Product(ProductID) NOT NULL,
```

C.
```
ProductID bigint DEFAULT
((NEXT VALUE FOR Production.ProductID_Seq OVER
(ORDER BY ManufacturingStepID))) NOT NULL,
```

D.
```
ProductID bigint DEFAULT
((NEXT VALUE FOR Production.ProductID_Seq OVER
(ORDER BY ManufacturingStepID)))
NOT NULL FOREIGN KEY REFERENCES
Production.Product(ProductID),
```

- ☐ Option A
- ☐ Option B
- ☐ Option C
- ☐ Option D

Question 33

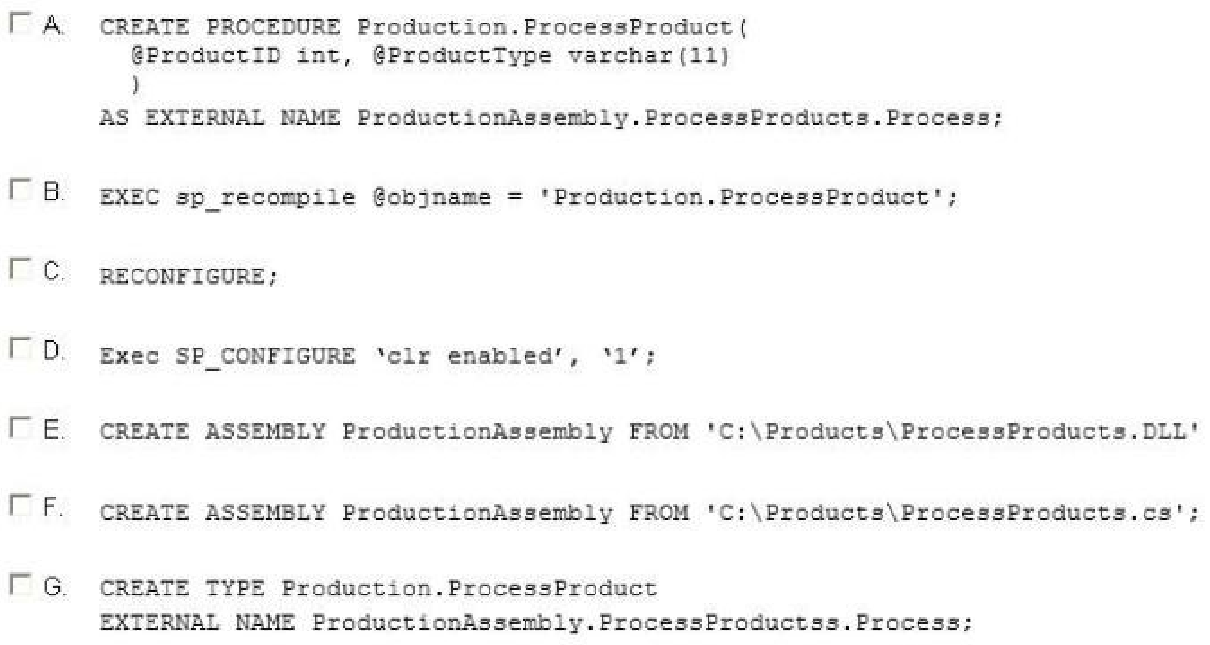
Which code segment should you use to define the ProductDetails column?

- ☐ ProductDetails xml (DOCUMENT Production.ProductDetailsSchema) NULL
- ☐ ProductDetails xml NULL
- ☐ ProductDetails xml (CONTENT Production.ProductDetailsSchema) NULL
- ☐ ProductDetails varchar(MAX) NULL

Question 34

You need to prepare the database to use the .NET Framework ProcessProducts component.

Which code segments should you execute? (Each correct answer presents part of the solution. Choose all that apply.)



☐ A.
```
CREATE PROCEDURE Production.ProcessProduct(
    @ProductID int, @ProductType varchar(11)
    )
AS EXTERNAL NAME ProductionAssembly.ProcessProducts.Process;
```

☐ B.
```
EXEC sp_recompile @objname = 'Production.ProcessProduct';
```

☐ C.
```
RECONFIGURE;
```

☐ D.
```
Exec SP_CONFIGURE ‘clr enabled’, ‘1’;
```

☐ E.
```
CREATE ASSEMBLY ProductionAssembly FROM 'C:\Products\ProcessProducts.DLL'
```

☐ F.
```
CREATE ASSEMBLY ProductionAssembly FROM 'C:\Products\ProcessProducts.cs';
```

☐ G.
```
CREATE TYPE Production.ProcessProduct
EXTERNAL NAME ProductionAssembly.ProcessProductss.Process;
```

- ☐ Option A
- ☐ Option B
- ☐ Option C
- ☐ Option D
- ☐ Option E
- ☐ Option F
- ☐ Option G

Question 35

While testing the CategoryFromType function, you discover that the function is returning 'Other'. You need to update CategoryFromType to return the category name.

Which line of code should you modify in CategoryFromType.sql?

- ☐ 04
- ☐ 05
- ☐ 12
- ☐ 14

Topic 5, Litware, Inc

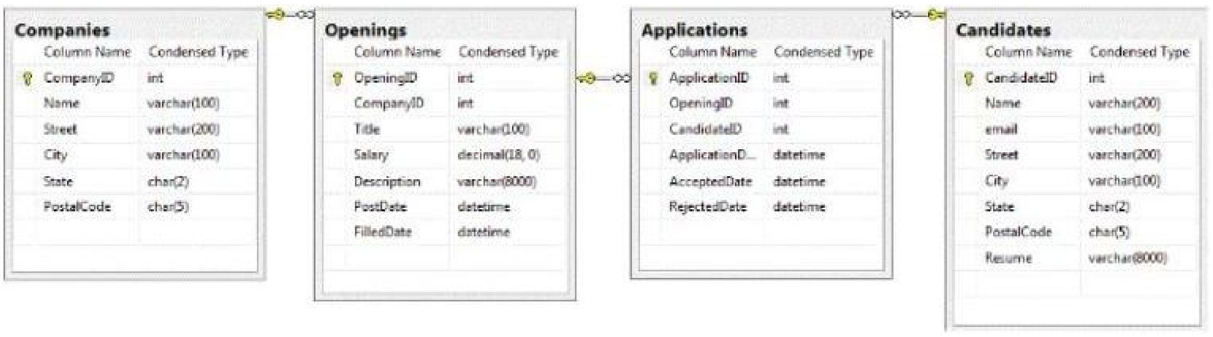
Case Study (5 questions)

Litware, Inc

General Overview

General Overview
You are a database developer for a company named Litware, Inc. Litware has a main office in Miami.

Litware has a job posting web application named WebApp1. WebApp1 uses a database named DB1. DB1 is hosted on a server named Server1. The database design of DB1 is shown in the exhibit. (Click the Exhibit button.)



Companies

Column Name	Condensed Type
CompanyID	int
Name	varchar(100)
Street	varchar(200)
City	varchar(100)
State	char(2)
PostalCode	char(5)

Openings

Column Name	Condensed Type
OpeningID	int
CompanyID	int
Title	varchar(100)
Salary	decimal(18, 0)
Description	varchar(8000)
PostDate	datetime
FilledDate	datetime

Applications

Column Name	Condensed Type
ApplicationID	int
OpeningID	int
CandidateID	int
ApplicationD...	datetime
AcceptedDate	datetime
RejectedDate	datetime

Candidates

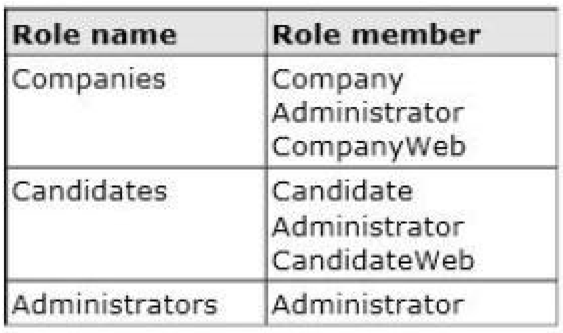
Column Name	Condensed Type
CandidateID	int
Name	varchar(200)
email	varchar(100)
Street	varchar(200)
City	varchar(100)
State	char(2)
PostalCode	char(5)
Resume	varchar(8000)

WebApp1 allows a user to log on as a job poster or a job seeker. Candidates can search for job openings based on keywords, apply to an opening, view their application, and load their resume in Microsoft Word format. Companies can add a job opening, view the list of candidates who applied to an opening, and mark an application as denied.

Users and Roles

DB1 has five database users named Company, CompanyWeb, Candidate, CandidateWeb, and Administrator.

DB1 has three user-defined database roles. The roles are configured as shown in the following table.



Role name	Role member
Companies	Company Administrator CompanyWeb
Candidates	Candidate Administrator CandidateWeb
Administrators	Administrator

Keyword Search

The keyword searches for the job openings are performed by using the following stored procedure named usp_GetOpenings:

```
CREATE PROCEDURE usp_GetOpenings
  @keyword varchar(max),
  @minsalary decimal(18,0) = 0
AS
DECLARE @plural varchar(max);
DECLARE @ing varchar(max);
SET @plural = @keyword + 's';
SET @ing = @keyword + 'ing';
SELECT o.Title, o.Salary, c.Name, o.Description
FROM Openings o
INNER JOIN Companies c ON c.CompanyID = o.CompanyID
WHERE (o.Description LIKE '%'+@keyword+'%'
  OR o.Description LIKE '%'+@plural+'%'
  OR o.Description LIKE '%'+@ing+'%')
  AND o.Salary >= @minsalary;
```

Opening Update

Updates to the Openings table are performed by using the following stored procedure named usp_UpdateOpening:

```
CREATE PROCEDURE usp_UpdateOpening
  @openingID int,
  @title varchar(100),
  @salary decimal(18,0),
  @description varchar(8000)
AS
UPDATE Openings
SET Title = @title,
  Salary = @salary,
  Description = @description
WHERE OpeningID = @openingID;
```

Problems and Reported Issues

Concurrency Problems
You discover that deadlocks frequently occur.

You identify that a stored procedure named usp_AcceptCandidate and a stored procedure named usp_UpdateCandidate generate deadlocks. The following is the code for usp_AcceptCandidate:

```
CREATE PROCEDURE usp_AcceptCandidate
  @applicationID int
AS
  DECLARE @date datetime;
  SET @date = GETDATE();
  UPDATE Applications
    SET AcceptedDate = @date
    WHERE ApplicationID = @applicationID;
  SELECT Name, email
    FROM Candidates c
    INNER JOIN Applications a
      ON a.CandidateID = c.CandidateID
    WHERE a.AcceptedDate IS NOT NULL;
```

Salary Query Issues
Users report that when they perform a search for job openings without specifying a minimum salary, only job openings that specify a minimum salary are displayed.

Log File Growth Issues
The current log file for DB1 grows constantly. The log file fails to shrink even when the daily SQL Server Agent Shrink Database task runs.

Performance Issues
You discover that a stored procedure named usp_ExportOpenings takes a long time to run and executes a table scan when it runs.

You also discover that the usp_GetOpenings stored procedure takes a long time to run and that the non-clustered index on the Description column is not being used.

Page Split Issues
On DB1, many page splits per second spike every few minutes.

Requirements

Security and Application Requirements
Litware identifies the following security and application requirements:
Only the Administrator, Company, and CompanyWeb database users must be able to execute the usp_UpdateOpening stored procedure.
Changes made to the database must not affect WebApp1.
Locking Requirements
Litware identifies the following locking requirements:
The usp_GetOpenings stored procedure must not be blocked by the usp_UpdateOpening stored procedure.
If a row is locked in the Openings table, usp_GetOpenings must retrieve the latest version of the row, even if the row was not committed yet.
Integration Requirements
Litware exports its job openings to an external company as XML data. The XML data uses the following format:

```
<Opening title="web programmer" salary="75000">
  This is the description of the opening
</Opening>
```

A stored procedure named usp_ExportOpenings will be used to generate the XML data. The following is the code for usp_ExportOpenings:

```sql
CREATE PROCEDURE usp_ExportOpenings
  @lastPost datetime
AS
SELECT Description
  , Title
  , Salary
FROM Openings
WHERE PostDate > @lastPost
  AND FilledDate IS NULL
```

The stored procedure will be executed by a SQL Server Integration Services (SSIS) package named Package1.

The XML data will be written to a secured folder named Folder1. Only a dedicated Active Directory account named Account1 is assigned the permissions to read from or write to Folder1.

Refactoring Requirements
Litware identifies the following refactoring requirements:
New code must be written by reusing the following query:

```sql
SELECT Title, Salary, Description
FROM Openings
WHERE Salary >= @minsalary
  AND FilledData IS NULL
```

The results from the query must be able to be joined to other queries.
Upload Requirements
Litware requires users to upload their job experience in a Word file by using WebApp1. WebApp1 will send the Word file to DB1 as a stream of bytes. DB1 will then convert the Word file to text before the contents of the Word file is saved to the Candidates table.

A database developer creates an assembly named Conversions that contains the following:
A class named Convert in the SqlConversions namespace
A method named ConvertToText in the Convert class that converts Word files to text

The ConvertToText method accepts a stream of bytes and returns text. The method is used in the following stored procedure:

```sql
CREATE PROCEDURE usp_UpdateCandidate
  @candidateID int,
  @wordResume varbinary(max)
AS
DECLARE @textResume varchar(8000);
SET @textResume = ConvertToText(@wordResume);
UPDATE Candidates SET Resume = @textResume
  WHERE CandidateID = @candidateID;
SELECT OpeningID, ApplicationDate
FROM Applications
WHERE CandidateID = @candidateID;
```

Job Application Requirements
A candidate can only apply to each job opening once.

Data Recovery Requirements
All changes to the database are performed by using stored procedures. WebApp1 generates a unique transaction ID for every stored procedure call that the application makes to the database.

If a server fails, you must be able to restore data to a specified transaction.

Question 42

You need to implement a solution that meets the security requirements.

Which statement should you execute?

A.
```
REVOKE EXEC ON usp_UpdateOpening FROM Candidates;
```

B.
```
DENY EXEC ON usp_UpdateOpening TO Candidates;
```

C.
```
ALTER PROCEDURE usp_UpdateOpening
@openingIDint,
@titlevarchar(100),
@salarydecimal(18,0),
@descriptionvarchar(8000)
WITH EXECUTE AS Administrator
AS
...
```

D.
```
ALTER PROCEDURE usp_UpdateOpening
@openingIDint,
@titlevarchar(100),
@salarydecimal(18,0),
@descriptionvarchar(8000)
WITH EXECUTE AS Company
AS
...
```

- ☐ Option A
- ☐ Option B
- ☐ Option C
- ☐ Option D

Question 43

You need to resolve the performance issues of the usp_ExportOpenings stored procedure. The solution must minimize the amount of hard disk space used.

Which statement should you execute on DB1?

- ☐ EXEC sp_dboption 'DB1', 'auto create statistics', 'TRUE';
- ☐ CREATE INDEX IX_Exp_Openings ON Openings(PostDate, FilledDate) INCLUDE (Description, Title, Salary);
- ☐ CREATE INDEX IX_Exp_Openings ON Openings(PostDate) INCLUDE (Description, Title, Salary) WHERE FilledDate IS NULL;
- ☐ EXEC sp_recompile 'usp_ExportOpenings';

Question 44

You need to recommend a solution that meets the concurrency problems.

What should you include in the recommendation?

- ☐ Modify the stored procedures to use the SERIALIZABLE isolation level.
- ☐ Modify the order in which usp_AcceptCandidate accesses the Applications table and the Candidates table.
- ☐ Modify the order in which usp_UpdateCandidate accesses the Applications table and the Candidates table.
- ☐ Modify the stored procedures to use the REPEATABLE READ isolation level.

Question 45

You need to resolve the performance issues of the usp_getOpenings stored procedure.

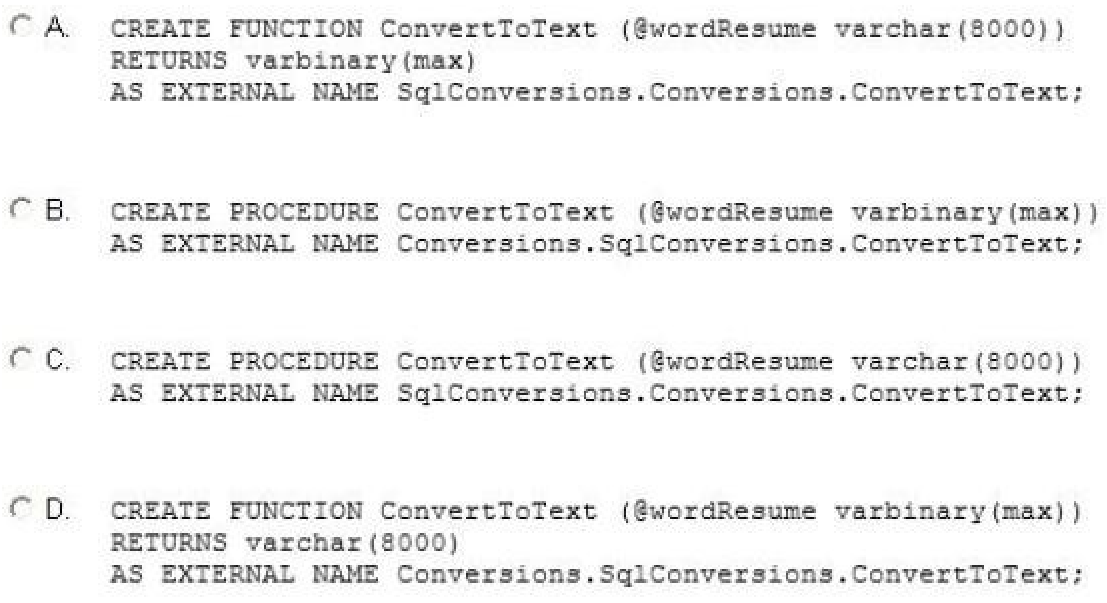
Which three actions should you perform? Each correct answer presents part of the solution.

- ☐ Delete lines 05 through 08
- ☐ Replace lines 12, 13, and 14 with the Transact-SQL segment:
- ☐ WHERE (CONTAINS(o.Description, 'ISABOUT(' +@keyword+' weight (.5))'))
- ☐ Create a full text index on the Description column
- ☐ Replace lines 12, 13, and 14 with the Transact_SQL segment:
- ☐ WHERE (CONTAINS(o.Description, @keyword))
- ☐ Replace lines 12, 13, and 14 with the Transact SQL Segment:
- ☐ WHERE (Contains(o.Description, 'FORMSOF(INFLECTIONAL, '+@keyword+')'))

Question 46

You need to implement a solution that addresses the upload requirements.

Which code segment should you use to implement the Conversions assembly?



A.
```
CREATE FUNCTION ConvertToText (@wordResume varchar(8000))
RETURNS varbinary(max)
AS EXTERNAL NAME SqlConversions.Conversions.ConvertToText;
```

B.
```
CREATE PROCEDURE ConvertToText (@wordResume varbinary(max))
AS EXTERNAL NAME Conversions.SqlConversions.ConvertToText;
```

C.
```
CREATE PROCEDURE ConvertToText (@wordResume varchar(8000))
AS EXTERNAL NAME SqlConversions.Conversions.ConvertToText;
```

D.
```
CREATE FUNCTION ConvertToText (@wordResume varbinary(max))
RETURNS varchar(8000)
AS EXTERNAL NAME Conversions.SqlConversions.ConvertToText;
```

- ☐ Option A
- ☐ Option B
- ☐ Option C
- ☐ Option D

Topic 6, Coho Winery

Case Study (5 questions)

Coho Winery

Overview

Overview
You are a database developer for a company named Coho Winery. Coho Winery has an office in London.

Coho Winery has an application that is used to process purchase orders from customers and retailers in 10 different countries.

The application uses a web front end to process orders from the Internet. The web front end adds orders to a database named Sales. The Sales database is managed by a server named Server1.

An empty copy of the Sales database is created on a server named Server2 in the London office. The database will store sales data for customers in Europe.

A new version of the application is being developed. In the new version, orders will be placed either by using the existing web front end or by loading an XML file.

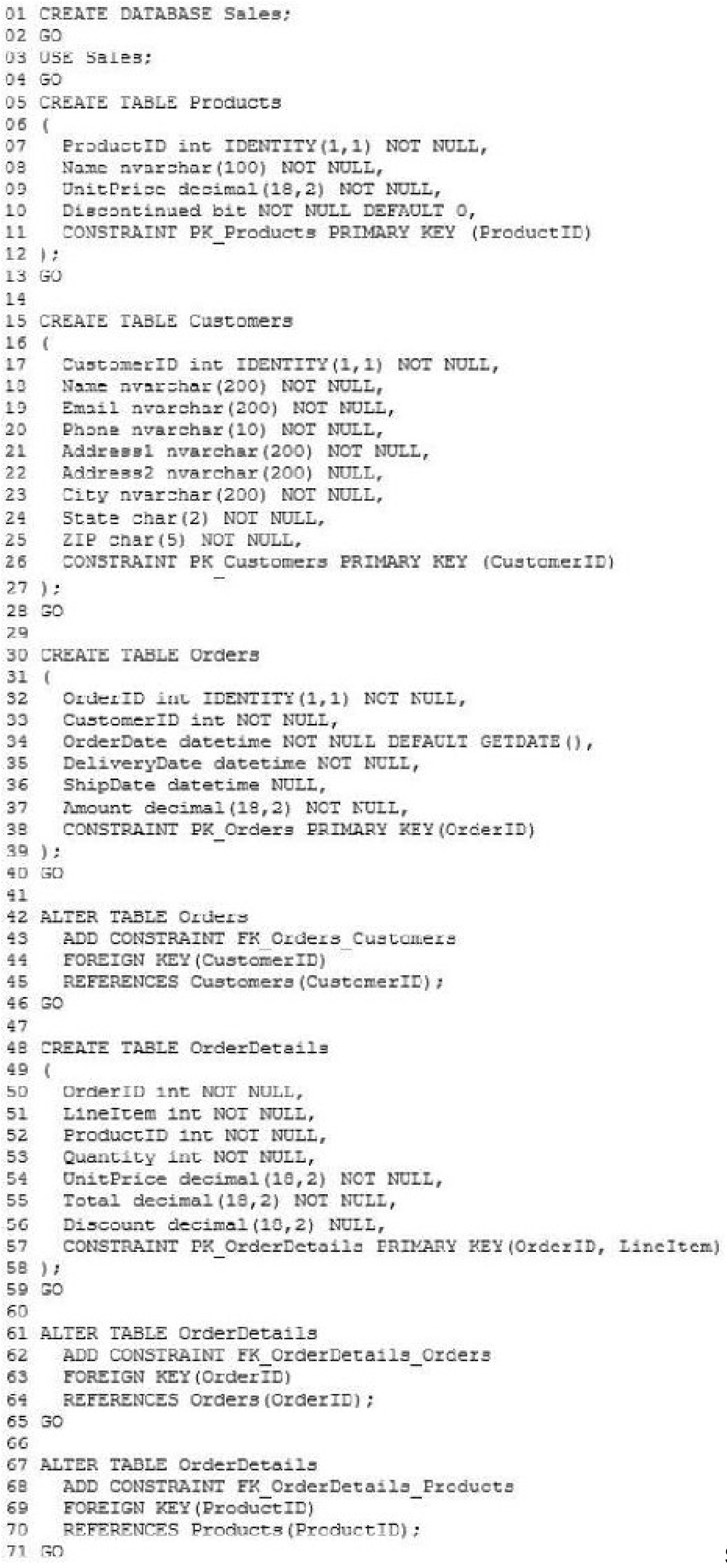
Once a week, you receive two files that contain the purchase orders and the order details of orders from offshore facilities.

You run the usp_ImportOders stored procedure and the usp_ImportOrderDetails stored procedure to copy the offshore facility orders to the Sales database.

The Sales database contains a table named Orders that has more than 20 million rows.

Database Definitions

Database and Tables
The following scripts are used to create the database and its tables:



```
CREATE DATABASE Sales;
GO
USE Sales;
GO
CREATE TABLE Products
(
  ProductID int IDENTITY(1,1) NOT NULL,
  Name nvarchar(100) NOT NULL,
  UnitPrice decimal(18,2) NOT NULL,
  Discontinued bit NOT NULL DEFAULT 0,
  CONSTRAINT PK_Products PRIMARY KEY (ProductID)
);
GO

CREATE TABLE Customers
(
  CustomerID int IDENTITY(1,1) NOT NULL,
  Name nvarchar(200) NOT NULL,
  Email nvarchar(200) NOT NULL,
  Phone nvarchar(10) NOT NULL,
  Address1 nvarchar(200) NOT NULL,
  Address2 nvarchar(200) NULL,
  City nvarchar(200) NOT NULL,
  State char(2) NOT NULL,
  ZIP char(5) NOT NULL,
  CONSTRAINT PK_Customers PRIMARY KEY (CustomerID)
);
GO

CREATE TABLE Orders
(
  OrderID int IDENTITY(1,1) NOT NULL,
  CustomerID int NOT NULL,
  OrderDate datetime NOT NULL DEFAULT GETDATE(),
  DeliveryDate datetime NOT NULL,
  ShipDate datetime NULL,
  Amount decimal(18,2) NOT NULL,
  CONSTRAINT PK_Orders PRIMARY KEY(OrderID)
);
GO

ALTER TABLE Orders
  ADD CONSTRAINT FK_Orders_Customers
  FOREIGN KEY(CustomerID)
  REFERENCES Customers(CustomerID);
GO

CREATE TABLE OrderDetails
(
  OrderID int NOT NULL,
  LineItem int NOT NULL,
  ProductID int NOT NULL,
  Quantity int NOT NULL,
  UnitPrice decimal(18,2) NOT NULL,
  Total decimal(18,2) NOT NULL,
  Discount decimal(18,2) NULL,
  CONSTRAINT PK_OrderDetails PRIMARY KEY(OrderID, LineItem)
);
GO

ALTER TABLE OrderDetails
  ADD CONSTRAINT FK_OrderDetails_Orders
  FOREIGN KEY(OrderID)
  REFERENCES Orders(OrderID);
GO

ALTER TABLE OrderDetails
  ADD CONSTRAINT FK_OrderDetails_Products
  FOREIGN KEY(ProductID)
  REFERENCES Products(ProductID);
GO
```

The following are the definitions of the stored procedures used in the database:

```
01 CREATE PROCEDURE usp_AddOrder
02   @customerID int,
03   @deliveryDate datetime,
04   @items ItemsTable READONLY,
05   @orderID int OUTPUT
06 AS
07 SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
08 BEGIN TRANSACTION;
09   DECLARE @amount decimal(18,2);
10   SELECT @amount = SUM(Quantity * UnitPrice) FROM @items;
11   INSERT INTO Orders (CustomerID, DeliveryDate, Amount)
12     VALUES (@customerID, @deliveryDate, @amount);
13   SELECT @orderID = @@IDENTITY;
14   INSERT INTO OrderDetails
15     SELECT @orderID, LineItem, ProductID, Quantity,
16       UnitPrice, Total, Discount
17     FROM @items;
18 COMMIT TRANSACTION;
19 GO

20 CREATE PROCEDURE usp_AddXMLOrder
21   @customerID int,
22   @deliverDate datetime,
23
24   @orderID int OUTPUT
25 AS
26 SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
27 BEGIN TRANSACTION;
28   DECLARE @itemsTable ItemsTable;
29   EXEC usp_ValidateAndGetItems @schema, @items, @itemsTable;
30   UPDATE Orders SET originalOrder = @items
31     WHERE OrderID = @orderID;
32 COMMIT TRANSACTION;
33 GO
34 CREATE PROCEDURE usp_ValidateAndGetItems
35   @schema sysname,
36   @items XML,
37   @itemsTable ItemsTable Output
38 AS

39 CREATE PROCEDURE usp_GetOrdersAndItems
40 AS
41
42   SELECT o.OrderID, o.OrderDate, o.DeliveryDate, o.Amount,
43          od.LineItem, od.Quantity, od.UnitPrice, p.Name
44   FROM Orders o
45   INNER JOIN OrderDetails od ON od.OrderID=o.OrderID
46   INNER JOIN Products p ON p.ProductID=od.ProductID
47   WHERE o.ShipDate IS NULL
48     AND o.DeliveryDate >= GETDATE() - 30;
49 GO
```

```
50 CREATE PROCEDURE usp_GetOrders
51 AS
52   SELECT OrderID, DeliveryDate, Amount
53   FROM Orders
54   WHERE ShipDate IS NULL
55   ORDER BY DeliveryDate;
56 GO
57
58 CREATE PROCEDURE usp_GetOrdersByProduct
59   @productID int
60
61 AS
62 SELECT OrderID, LineItem, Quantity,
63   UnitPrice, Total, Discount
64 FROM OrderDetails
65
66 WHERE ProductID = @productID;
67 GO
68
69 CREATE PROCEDURE usp_ImportOrders
70 AS
71 BULK INSERT Orders
72   FROM 'f:\orders\orders.tbl'
73   WITH
74     (
75       FIELDTERMINATOR =' |',
76       ROWTERMINATOR =' |\n'
77     );
78 GO

79 CREATE PROCEDURE usp_ImportOrderDetails
80   @firstRow int
81 AS
82 BULK INSERT OrderDetails
83   FROM 'f:\orders\details.tbl'
84   WITH
85     (
86
87       FIRSTROW = @firstRow,
88       FIELDTERMINATOR =' |',
89       ROWTERMINATOR =' |\n'
90     );
91 GO
```

Indexes

The following indexes are part of the Sales database:

```
01 CREATE INDEX IX_Orders_ShipDate
02   ON Orders(Shipdate)
03
04   INCLUDE (CustomerID, OrderDate, Amount);
05 GO
```

Data Import

The XML files will contain the list of items in each order. Each retailer will have its own XML schema and will be able to use different types of encoding. Each XML schema will use a default namespace. The default namespaces are not guaranteed to be unique.

For testing purposes, you receive an XSD file from a customer.

For testing purposes, you also create an XML schema collection named ValidateOrder. ValidateOrder contains schemas for all of the retailers.

The new version of the application must validate the XML file, parse the data, and store the parsed data along with the original XML file in the database. The original XML file must be stored without losing any data.

Reported Issues

Performance Issues
You notice the following for the usp_GetOrdersAndItems stored procedure:
The stored procedure takes a long time to complete.
Less than two percent of the rows in the Orders table are retrieved by usp_GetOrdersAndItems.
A full table scan runs when the stored procedure executes.
The amount of disk space used and the amount of time required to insert data are very high.

You notice that the usp_GetOrdersByProduct stored procedure uses a table scan when the stored procedure is executed.

Page Split Issues
Updates to the Orders table cause excessive page splits on the IX_Orders_ShipDate index.

Requirements

Site Requirements
Users located in North America must be able to view sales data for customers in North America and Europe in a single report. The solution must minimize the amount of traffic over the WAN link between the offices.

Bulk Insert Requirements
The usp_ImportOrderDetails stored procedure takes more than 10 minutes to complete. The stored procedure runs daily. If the stored procedure fails, you must ensure that the stored procedure restarts from the last successful set of rows.

Index Monitoring Requirements
The usage of indexes in the Sales database must be monitored continuously. Monitored data must be maintained if a server restarts. The monitoring solution must minimize the usage of memory resources and processing resources.

Question 56

You need to ensure that usp_AddXMLOrder can be used to validate the XML input from the retailers. Which parameters should you add to usp_AddXMLOrder on line 04 and line 05? (Each correct answer presents part of the solution. Choose all that apply.)

- ☐ @schema varbinary(100).
- ☐ @items varchar(max).

- ☐ @schema sysname.
- ☐ @items varbinary(max).
- ☐ @items xml.
- ☐ @schema xml.

Question 57

You need to ensure that a new execution plan is used by usp_GetOrdersByProduct each time the stored procedure runs.

What should you do?

- ☐ Execute sp_help usp_GetOrdersByProduct\
- ☐ Add WITH (FORCESEEK) to line 69 in usp.GetOrdersByProduct.
- ☐ Add WITH RECOMPILE to line 64 in usp.GetOrdersByProduct.
- ☐ Execute sp_recompile usp.GetOrdersByProduct'.

Question 58

You need to modify usp_GetOrdersAndItems to ensure that an order is NOT retrieved by usp_GetOrdersAndItems while the order is being updated.

What should you add to usp_GetOrdersAndItems?

- ☐ Add SET TRANSACTION ISOLATION LEVEL SERIALIZABLE to line 03.
- ☐ Add SET TRANSACTION ISOLATION LEVEL SNAPSHOT to line 03.
- ☐ Add (UPDLOCK) to the end of line 06.
- ☐ Add (READPAST) to the end of line 06.

Question 59

You need to modify the Orders table to store the XML data used by the retailers.

Which statement should you execute?

- ☐ ALTER Orders ADD originalOrder XML (ValidateOrder);
- ☐ ALTER Orders ADD originalOrder XML;
- ☐ ALTER Orders ADD originalOrder varchar(max);
- ☐ ALTER Orders ADD originalOrder varbinary(max);

Question 60

You need to implement a solution that meets the site requirements.

What should you implement?

- ☐ an indexed view on Server1
- ☐ a distributed view on Server1
- ☐ a distributed view on Server2
- ☐ an indexed view on Server2

Topic 7, Fourth Coffee

Case Study (5 questions)

Fourth Coffee

Background

Corporate Information
Fourth Coffee is global restaurant chain. There are more than 5,000 locations worldwide.

Physical Locations

Currently a server at each location hosts a SQL Server 2012 instance. Each instance contains a database called StoreTransactions that stores all transactions from point of sale and uploads summary batches nightly.

Each server belongs to the COFFECORP domain. Local computer accounts access the StoreTransactions database at each store using sysadmin and datareaderwriter roles.

Planned changes

Fourth Coffee has three major initiatives:
The IT department must consolidate the point of sales database infrastructure.
The marketing department plans to launch a mobile application for micropayments.
The finance department wants to deploy an internal tool that will help detect fraud.

Initially, the mobile application will allow customers to make micropayments to buy coffee and other items on the company web site. These micropayments may be sent as gifts to other users and redeemed within an hour of ownership transfer. Later versions will generate profiles based on customer activity that will push texts and ads generated by an analytics application.

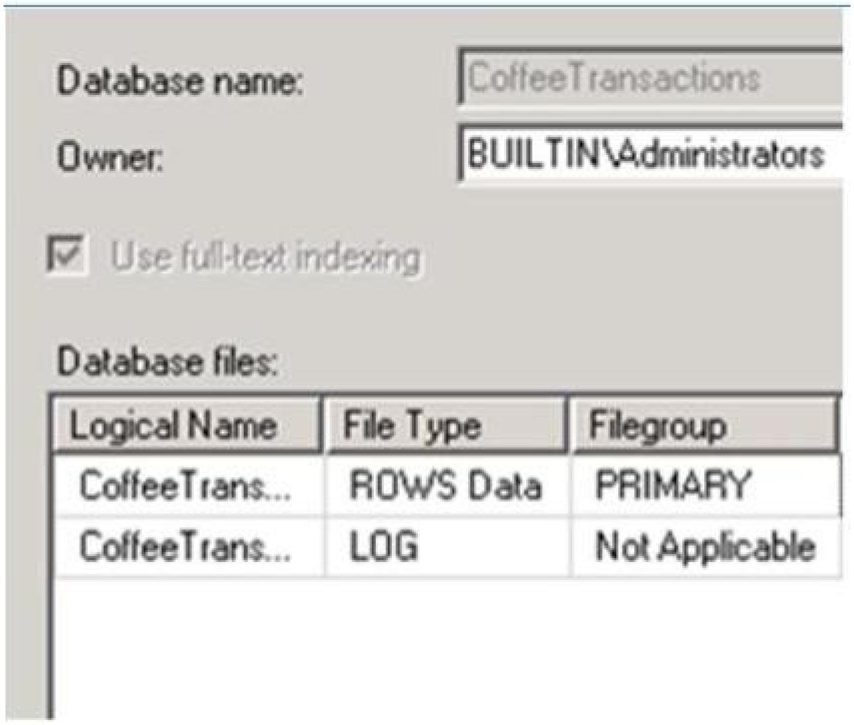
When the consolidation is finished and the mobile application is in production, the micropayments and point of sale transactions will use the same database.

Existing Environment

Existing Application Environment
Some stores have been using several pilot versions of the micropayment application. Each version currently is in a database that is independent from the point of sales systems. Some versions have been used in field tests at local stores, and others are hosted at corporate servers. All pilot versions were developed by using SQL Server 2012.

Existing Support Infrastructure

The proposed database for consolidating micropayments and transactions is called CoffeeTransactions. The database is hosted on a SQL Server 2014 Enterprise Edition instance and has the following file structures:



Database name: CoffeeTransactions

Owner: BUILTIN\Administrators

☑ Use full-text indexing

Database files:

Logical Name	File Type	Filegroup
CoffeeTrans...	ROWS Data	PRIMARY
CoffeeTrans...	LOG	Not Applicable

Business Requirements

General Application Solution Requirements
The database infrastructure must support a phased global rollout of the micropayment application and consolidation.

The consolidated micropayment and point of sales database will be into a CoffeeTransactions database. The infrastructure also will include a new CoffeeAnalytics database for reporting on content from CoffeeTransactions.

Mobile applications will interact most frequently with the micropayment database for the following activities:
Retrieving the current status of a micropayment;
Modifying the status of the current micropayment; and
Canceling the micropayment.

The mobile application will need to meet the following requirements:
Communicate with web services that assign a new user to a micropayment by using a stored procedure named usp_AssignUser.
Update the location of the user by using a stored procedure named usp_AddMobileLocation.

The fraud detection service will need to meet the following requirements:
Query the current open micropayments for users who own multiple micropayments by using a stored procedure named usp.LookupConcurrentUsers.

Persist the current user locations by using a stored procedure named usp_MobileLocationSnapshot.
Look at the status of micropayments and mark micropayments for internal investigations.
Move micropayments to dbo.POSException table by using a stored procedure named ups_DetectSuspiciousActivity.
Detect micropayments that are flagged with a StatusId value that is greater than 3 and that occurred within the last minute.

The CoffeeAnalytics database will combine imports of the POSTransaction and MobileLocation tables to create a UserActivity table for reports on the trends in activity. Queries against the UserActivity table will include aggregated calculations on all columns that are not used in filters or groupings.

Micropayments need to be updated and queried for only a week after their creation by the mobile application or fraud detection services.

Performance
The most critical performance requirement is keeping the response time for any queries of the POSTransaction table predictable and fast.
Web service queries will take a higher priority in performance tuning decisions over the fraud detection agent queries.

Scalability
Queries of the user of a micropayment cannot return while the micropayment is being updated, but can show different users during different stages of the transaction.

The fraud detection service frequently will run queries over the micropayments that occur over different time periods that range between 30 seconds and ten minutes.

The POSTransaction table must have its structure optimized for hundreds of thousands of active micropayments that are updated frequently.

All changes to the POSTransaction table will require testing in order to confirm the expected throughput that will support the first year's performance requirements.

Updates of a user's location can tolerate some data loss. Initial testing has determined that the POSTransaction and POSException tables will be migrated to an in-memory optimized table.

Availability
In order to minimize disruption at local stores during consolidation, nightly processes will restore the databases to a staging server at corporate headquarters.

Technical Requirements

Security
The sensitive nature of financial transactions in the store databases requires certification of the COFFECORP\Auditors group at corporate that will perform audits of the data. Members of the COFFECORP\Auditors group cannot have sysadmin or datawriter access to the database. Compliance requires that the data stewards have access to any restored StoreTransactions database without changing any security settings at a database level.

Nightly batch processes are run by the services account in the COFFECORP\StoreAgent group and need to be able to restore and verify the schema of the store databases match.

No Windows group should have more access to store databases than is necessary.

Maintainability
You need to anticipate when POSTransaction table will need index maintenance.

When the daily maintenance finishes, micropayments that are one week old must be available for queries in UserActivity table but will be queried most frequently within their first week and will require support for in-memory queries for data within first week.

The maintenance of the UserActivity table must allow frequent maintenance on the day's most recent activities with minimal impact on the use of disk space and the resources available to queries. The processes that add data to the UserActivity table must be able to update data from any time period, even while maintenance is running.

The index maintenance strategy for the UserActivity table must provide the optimal structure for both maintainability and query performance.

All micropayments queries must include the most permissive isolation level available for the maximum throughput.

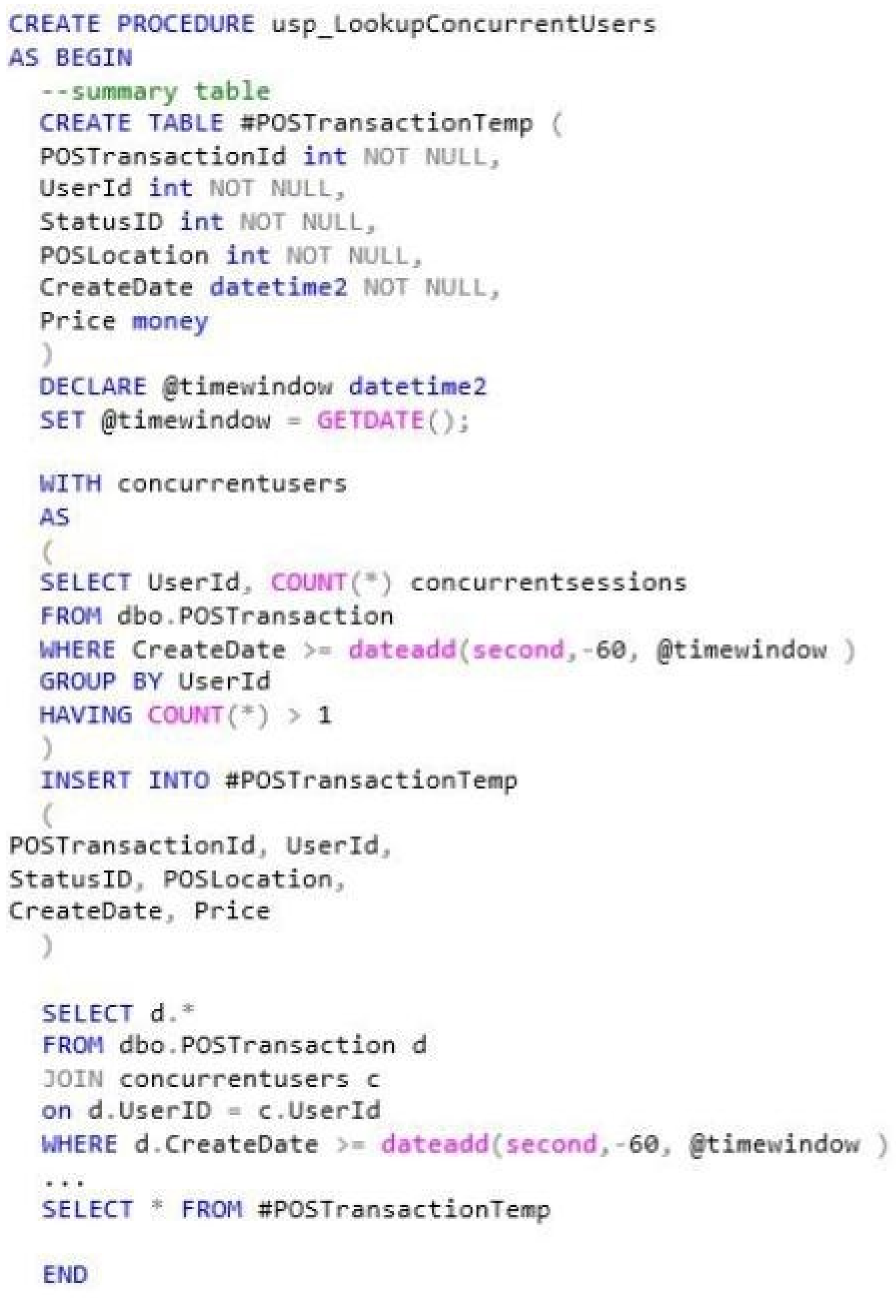
In the event of unexpected results, all stored procedures must provide error messages in text message to the calling web service.
Any modifications to stored procedures will require the minimal amount of schema changes necessary to increase the performance.

Performance
Stress testing of the mobile application on the proposed CoffeeTransactions database uncovered performance bottlenecks. The sys.dm_os_wait_stats Dynamic Management View (DMV) shows high wait_time values for WRTTELOG and PAGEIOLATCHJJP wait types when updating the MobileLocation table.

Updates to the MobileLocation table must have minimal impact on physical resources.

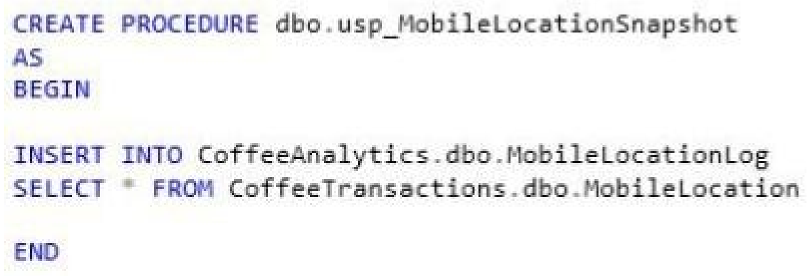
Supporting Infrastructure

The stored procedure usp_LookupConcurrentUsers has the current implementation:



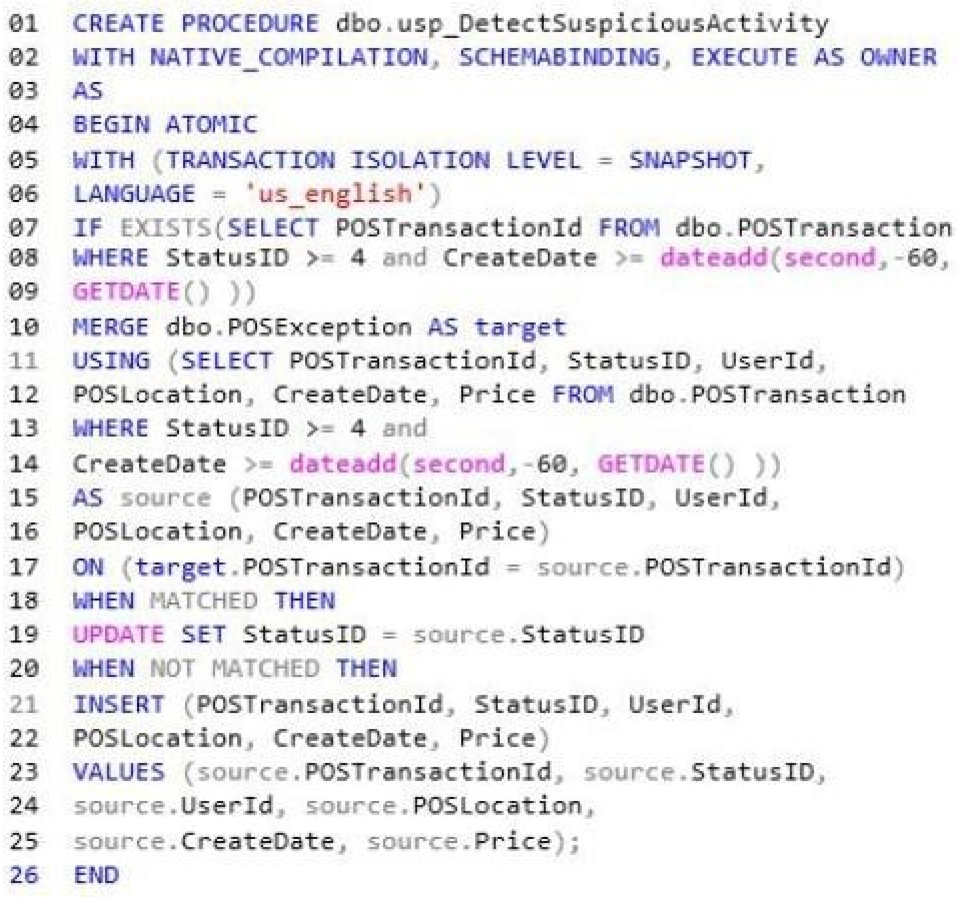
```sql
CREATE PROCEDURE usp_LookupConcurrentUsers
AS BEGIN
  --summary table
  CREATE TABLE #POSTransactionTemp (
  POSTransactionId int NOT NULL,
  UserId int NOT NULL,
  StatusID int NOT NULL,
  POSLocation int NOT NULL,
  CreateDate datetime2 NOT NULL,
  Price money
  )
  DECLARE @timewindow datetime2
  SET @timewindow = GETDATE();

  WITH concurrentusers
  AS
  (
  SELECT UserId, COUNT(*) concurrentsessions
  FROM dbo.POSTransaction
  WHERE CreateDate >= dateadd(second,-60, @timewindow )
  GROUP BY UserId
  HAVING COUNT(*) > 1
  )
  INSERT INTO #POSTransactionTemp
  (
POSTransactionId, UserId,
StatusID, POSLocation,
CreateDate, Price
  )

  SELECT d.*
  FROM dbo.POSTransaction d
  JOIN concurrentusers c
  on d.UserID = c.UserId
  WHERE d.CreateDate >= dateadd(second,-60, @timewindow )
  ...
  SELECT * FROM #POSTransactionTemp

  END
```

The current stored procedure for persisting a user location is defined in the following code:



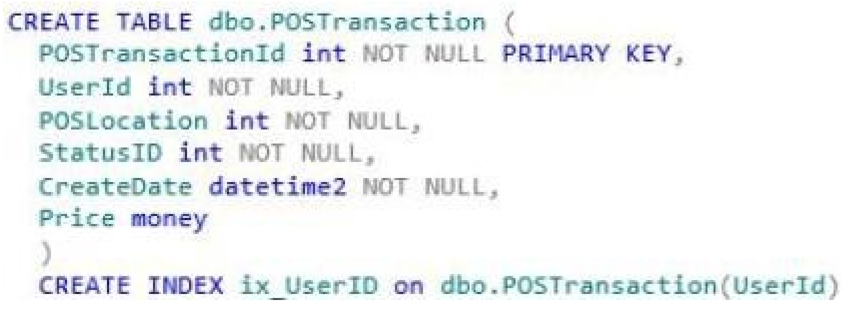
```sql
CREATE PROCEDURE dbo.usp_MobileLocationSnapshot
AS
BEGIN

INSERT INTO CoffeeAnalytics.dbo.MobileLocationLog
SELECT * FROM CoffeeTransactions.dbo.MobileLocation

END
```

The current stored procedure for managing micropayments needing investigation is defined in the following code:



```
CREATE PROCEDURE dbo.usp_DetectSuspiciousActivity
WITH NATIVE_COMPILATION, SCHEMABINDING, EXECUTE AS OWNER
AS
BEGIN ATOMIC
WITH (TRANSACTION ISOLATION LEVEL = SNAPSHOT,
LANGUAGE = 'us_english')
IF EXISTS(SELECT POSTransactionId FROM dbo.POSTransaction
WHERE StatusID >= 4 and CreateDate >= dateadd(second,-60,
GETDATE() ))
MERGE dbo.POSException AS target
USING (SELECT POSTransactionId, StatusID, UserId,
POSLocation, CreateDate, Price FROM dbo.POSTransaction
WHERE StatusID >= 4 and
CreateDate >= dateadd(second,-60, GETDATE() ))
AS source (POSTransactionId, StatusID, UserId,
POSLocation, CreateDate, Price)
ON (target.POSTransactionId = source.POSTransactionId)
WHEN MATCHED THEN
UPDATE SET StatusID = source.StatusID
WHEN NOT MATCHED THEN
INSERT (POSTransactionId, StatusID, UserId,
POSLocation, CreateDate, Price)
VALUES (source.POSTransactionId, source.StatusID,
source.UserId, source.POSLocation,
source.CreateDate, source.Price);
END
```

The current table, before implementing any performance enhancements, is defined as follows:



```
CREATE TABLE dbo.POSTransaction (
  POSTransactionId int NOT NULL PRIMARY KEY,
  UserId int NOT NULL,
  POSLocation int NOT NULL,
  StatusID int NOT NULL,
  CreateDate datetime2 NOT NULL,
  Price money
  )
  CREATE INDEX ix_UserID on dbo.POSTransaction(UserId)
```

Question 73

You need to optimize the index structure that is used by the tables that support the fraud detection services.

What should you do?

- ☐ Add a hashed nonclustered index to CreateDate.
- ☐ Add a not hash nonclustered index to CreateDate.
- ☐ Add a not hash clustered index on POSTransactionId and CreateDate.
- ☐ Add a hashed clustered index on POSTransactionId and CreateDate.

Question 74

You need to implement security for the restore and audit process. What should you do?

- ☐ Grant the COFFECORP\Auditors group ALTER ANY CONNECTION and SELECT ALL USER SECURABLES permissions. Grant the COFFECORP\StoreAgent group ALTER ANY CONNECTION and IMPERSONATE ANY LOGIN permissions.
- ☐ Grant the COFFECORP\Auditors group CONNECT ANY DATABASE and IMPERSONATE ANY LOGIN permissions. Grant the COFFECORP\StoreAgent group CONNECT ANY DATABASE and SELECT ALL USER SECURABLES permissions.
- ☐ Grant the COFFECORP\Auditors group ALTER ANY CONNECTION and IMPERSONATE ANY LOGIN permissions. Grant the COFFECORP\StoreAgent group ALTER ANY CONNECTION and SELECT ALL USER SECURABLES permissions.
- ☐ Grant the COFFECORP\Auditors group CONNECT ANY DATABASE and SELECT ALL USER SECURABLES permissions. Grant the COFFECORP\StoreAgent group CONNECT ANY DATABASE and IMPERSONATE ANY LOGIN permissions.

Question 75

You need to monitor the health of your tables and indexes in order to implement the required index maintenance strategy.

What should you do?

- ☐ Query system DMVs to monitor avg_chain_length and max_chain_length. Create alerts to notify you when these values converge.
- ☐ Create a SQL Agent alert when the File Table: Avg time per file I/O request value is increasing.
- ☐ Query system DMVs to monitor total_bucket_count. Create alerts to notify you when this value increases.
- ☐ Query system DMVs to monitor total_bucket_count. Create alerts to notify you when this value decreases.

Question 76

You need to modify the stored procedure usp_LookupConcurrentUsers.

What should you do?

- ☐ Add a clustered index to the summary table.
- ☐ Add a nonclustered index to the summary table.
- ☐ Add a clustered columnstore index to the summary table.
- ☐ Use a table variable instead of the summary table.

Question 77

You need to modify the stored procedure usp_LookupConcurrentUsers.

What should you do?

- ☐ Use the summary table as an in-memory optimized table with a non-hash clustered index.
- ☐ Use the summary table as an in-memory optimized table with a non-hash nonclustered index.
- ☐ Use a type variable instead of the summary table.
- ☐ Add a clustered index to the summary table.

Topic 8, Mix Questions

(5 questions)

Question 79

You execute the following code:

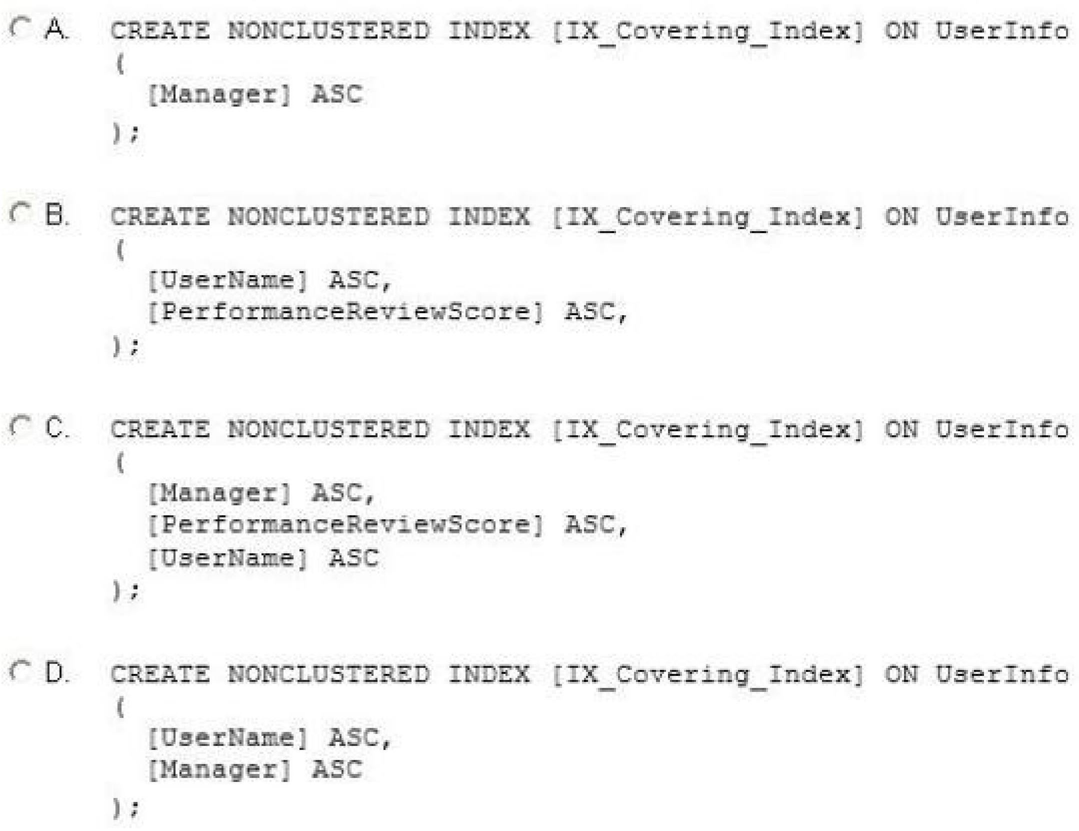
```
CREATE TABLE UserInfo
(
  ID int NOT NULL IDENTITY (1, 1)
  CONSTRAINT PK_UserInfo PRIMARY KEY CLUSTERED,
  UserName varchar(100) NOT NULL,
  Manager varchar(100) NULL,
  HireDate date NOT NULL,
  PerformanceReviewScore int NULL
);
```

You have a stored procedure that includes the following SELECT statement:

```
SELECT UserName, PerformanceReviewScore
FROM UserInfo
WHERE Manager = 'Ben Smith';
```

You need to create a covering index on UserInfo.

Which code segment should you execute?



A.
```
CREATE NONCLUSTERED INDEX [IX_Covering_Index] ON UserInfo
(
  [Manager] ASC
);
```

B.
```
CREATE NONCLUSTERED INDEX [IX_Covering_Index] ON UserInfo
(
  [UserName] ASC,
  [PerformanceReviewScore] ASC,
);
```

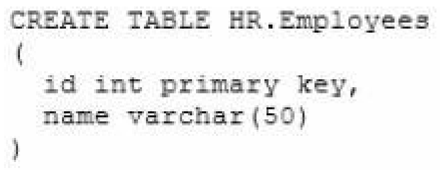
C.
```
CREATE NONCLUSTERED INDEX [IX_Covering_Index] ON UserInfo
(
  [Manager] ASC,
  [PerformanceReviewScore] ASC,
  [UserName] ASC
);
```

D.
```
CREATE NONCLUSTERED INDEX [IX_Covering_Index] ON UserInfo
(
  [UserName] ASC,
  [Manager] ASC
);
```

- ☐ Option A
- ☐ Option B
- ☐ Option C
- ☐ Option D

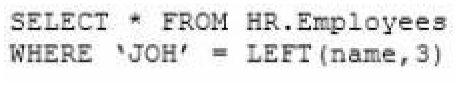
Question 80

You have a Microsoft SQL Azure database that contains a table named Employees.



```
CREATE TABLE HR.Employees
(
  id int primary key,
  name varchar(50)
)
```

You create a non-clustered index named EmployeeName on the name column.



```
SELECT * FROM HR.Employees
WHERE 'JOH' = LEFT(name,3)
```

You write the following query to retrieve all of the employees that have a name that starts with the letters JOH:

You discover that the query performs a table scan.

You need to ensure that the query uses EmployeeName.

What should you do?

- ☐ Recreate EmployeeName as a unique index
- ☐ Recreate EmployeeName as a clustered index
- ☐ Replace LEFT(name,3) = 'JOH' by using name like 'JOH%'
- ☐ Replace LEFT(name,3) = 'JOH' by using substring(name, 1, 3) = 'JOH'

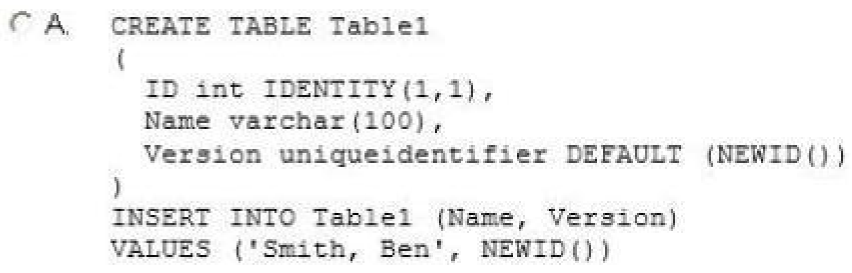
Question 81

You are creating a table to support an application that will cache data outside of SQL Server.

The application will detect whether cached values were changed before it updates the values.

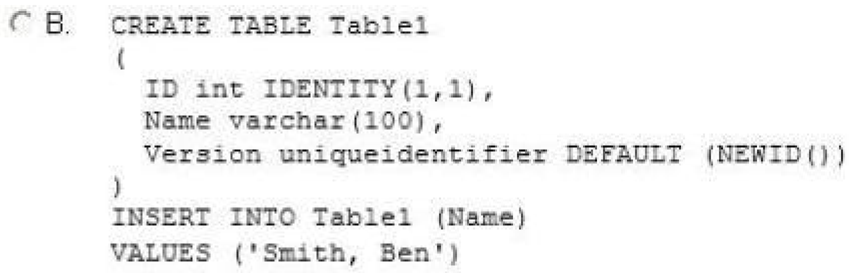
You need to create the table, and then verify that you can insert a row into the table.

Which code segment should you use?



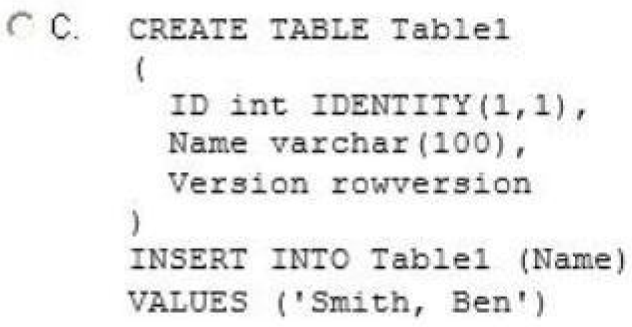
A.

```
CREATE TABLE Table1
(
  ID int IDENTITY(1,1),
  Name varchar(100),
  Version uniqueidentifier DEFAULT (NEWID())
)
INSERT INTO Table1 (Name, Version)
VALUES ('Smith, Ben', NEWID())
```



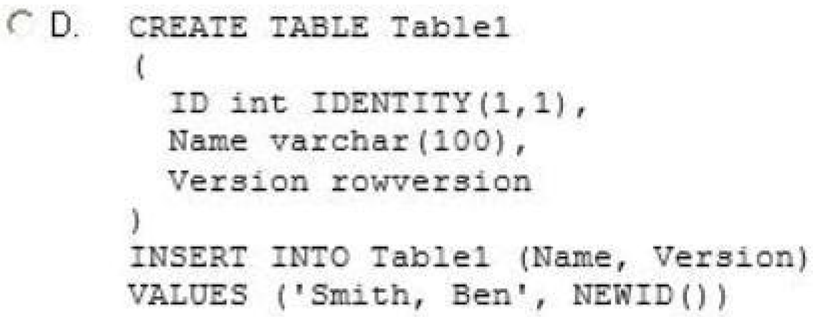
B.

```
CREATE TABLE Table1
(
  ID int IDENTITY(1,1),
  Name varchar(100),
  Version uniqueidentifier DEFAULT (NEWID())
)
INSERT INTO Table1 (Name)
VALUES ('Smith, Ben')
```



C.

```
CREATE TABLE Table1
(
  ID int IDENTITY(1,1),
  Name varchar(100),
  Version rowversion
)
INSERT INTO Table1 (Name)
VALUES ('Smith, Ben')
```



D.

```
CREATE TABLE Table1
(
  ID int IDENTITY(1,1),
  Name varchar(100),
  Version rowversion
)
INSERT INTO Table1 (Name, Version)
VALUES ('Smith, Ben', NEWID())
```

- ☐ Option A
- ☐ Option B
- ☐ Option C

- ☐ Option D

Question 82

You have a database that uses the following management views:
Sys.dm_os_volume_stats
Sys.dm_db_partition_stats
Sys.dm_db_file_space_usage
Sys.fulltext_indexes

You plan to migrate the database to Microsoft SQL Azure.

You need to identify which view can be used in SQL Azure.

Which view should you identify?

- ☐ sys.fulltext_indexes
- ☐ sys.dm_db_file_space_usage
- ☐ sys.dm_os_volume_stats
- ☐ sys.dm_db_partition_stats

Question 83

You discover a sudden increase in processor utilization on a server that has SQL Server installed.

You need to correlate server performance and database activity for an extended time period.

Which two tools should you use? Each correct answer presents part of the solution.

- ☐ Activity Monitor
- ☐ Performance Monitor
- ☐ SQL Server Profiler
- ☐ sp_who2
- ☐ SQL Server Extended Events