Microsoft _CertifyMe_ 70-451_v2010-05-25_181q_by-Bondo

Number: 70-451
Passing Score: 700
Time Limit: 120 min
File Version: 2010-05-25

Exam: Microsoft _CertifyMe 70-450

Version: 2010-05-25

Question : 181

Good luck everiwun!

by-Bondo

Exam A

QUESTION 1
You are designing a database that will store telephone numbers. You need to ensure that only phone numbers that use a specific format are written to the database.
What should you create?

A. a CHECK constraint
B. a computed column
C. a DEFAULT constraint
D. a persisted computed column

Answer: A
Section: (none)

Explanation/Reference:

QUESTION 2
You are a database developer on an instance of SQL Server 2008.
You need to provide a custom function that will perform complex calculations on groups of data in one of your tables. You want users to be able to reference the function directly from a query as if they were using a built-in scalar function. Which type of object should you create?

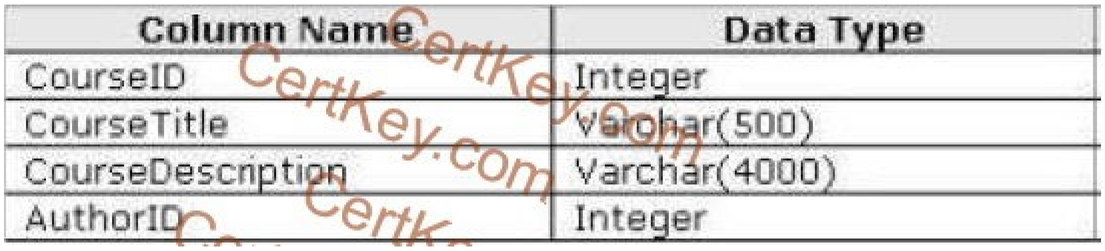
A. a CLR user-defined type
B. a Transact-SQL user-defined function that returns a scalar value
C. a CLR user-defined aggregate
D. a Transact-SQL user-defined stored procedure

Answer: C
Section: (none)

Explanation/Reference:
CLR user-defined aggregates allow you to create custom aggregate functions that users can call like other built-in aggregate functions, such as `SUM`, `MAX`, or `COUNT`. In a .NET language, you create the code necessary to implement the desired aggregation functionality and compile the class into an assembly. You should note that the class must meet specific requirements and implement specific methods to be able to use it for custom aggregation. After you create the assembly, you can register it in SQL Server using the `CREATE ASSEMBLY` statement. Then, you can use the `CREATE AGGREGATE` statement to create the custom aggregate function with the name users will use to access the custom aggregate function.
All of the other options are incorrect because to implement a custom aggregation, you must use a CLR user-defined aggregate.
CLR user-defined types allow you to create custom data types using a .NET Framework language, and then use these types from SQL Server. You can leverage the functionality of the .NET language that is not supported in SQL Server. You can also create a CLR user-defined type and then create a CLR user-defined aggregate that performs aggregations on a group of the specified type.

QUESTION 3
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. The full-text search component is installed in the database which supports a Web site. Look at the exhibit below:
You intend to create a table named Courses which has the structure above. On the basis of the CourseTitle field, users of the Web site will search for courses. When the search is launched for a course by a user, a full-

text must be constructed to ensure the compliances below: when the exact search phrase is found, rows are returned; rows are in order of how well they match with the search phrase. So what should you specify in the full-text query?



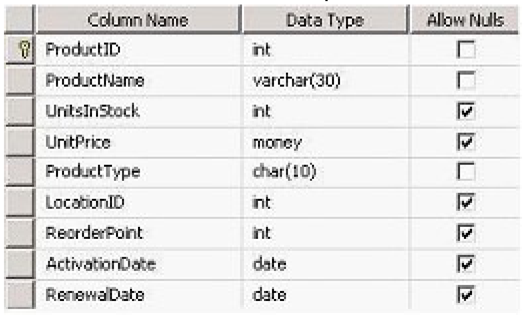
Column Name	Data Type
CourseID	Integer
CourseTitle	Varchar(500)
CourseDescription	Varchar(4000)
AuthorID	Integer

A. A CONTAINS predicate
B. A FREETEXT predicate
C. A CONTAINSTABLE function
D. A FREETEXTTABLE function

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 4
You are a database developer on an instance of SQL Server 2008. Your **Product** table is defined as follows:



	Column Name	Data Type	Allow Nulls
🔑	ProductID	int	☐
	ProductName	varchar(30)	☐
	UnitsInStock	int	✓
	UnitPrice	money	✓
	ProductType	char(10)	☐
	LocationID	int	✓
	ReorderPoint	int	✓
	ActivationDate	date	✓
	RenewalDate	date	✓

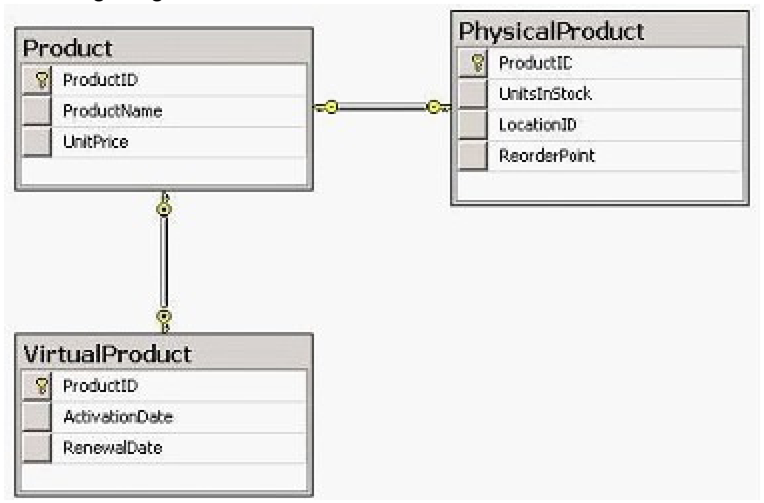
Your company sells two types of products identified by the **ProductType** column: physical and virtual. Physical products are stored in inventory locations. Virtual products, such as software downloads and subscriptions, are not stored in physical inventory locations.
You are developing an Entity Data Model (EDM) using the Entity Framework. You want to create the EDM to make virtual and physical products available while only exposing the necessary attributes for each.
Which data model should you create?

A. a data model that implements Table-per-Hierarchy inheritance
B. a data model that implements Table-per-Type inheritance
C. a data model that includes a complex type
D. a data model that implements a single entity with multiple associations

Answer: A
Section: (none)

Explanation/Reference:
The Table-per-Hierarchy inheritance model allows you to create an EDM that supports inheritance within a single database table. In this scenario, you have a **Product** table that contains a **ProductType** column. The

ProductType column, known as the discriminator column, identifies whether the product is a physical product or a virtual product. Certain columns in the **Product** table are applicable to physical products but not to virtual products. For example, a physical product will have a value for the **UnitsInStock**, **LocationID**, and **ReorderPoint** columns, and a virtual product will not.
To implement such a model in this scenario, you would create a base entity to represent a **Product**. Then you would create additional entities, for example **PhysicalProduct** and **VirtualProduct**, which inherit from the base entity. The **PhysicalProduct** entity would contain the properties applicable to products in physical inventory, and the **VirtualProduct** entity would contain the properties applicable to virtual products. You should note that because the **UnitsInStock**, **LocationID**, and **ReorderPoint** columns would not be applicable to virtual products and the **ActivationDate** and **RenewalDate** columns would not be applicable to physical products, all of these columns must be defined to allow null values.
You should not create a data model that implements Table-per-Type inheritance. The Table-per-Type inheritance model allows you to create an EDM that supports inheritance with multiple tables. You would use this inheritance model if you had a **Product** table that contained columns relating to all products and other tables that contained columns relating to specific product types. For example, you might have a **Product** table that has a one-to-one relationship with both the **PhysicalProduct** and **VirtualProduct** tables, as shown in the following diagram:



The **Product** table would hold the columns related to all products, and the **PhysicalProduct** and **VirtualProduct** tables would contain columns of their respective product types. Each of these tables would have to be represented by a separate entity in the data model. The **Product** entity would be the base entity, and the **PhysicalProduct** and **VirtualProduct** entities would inherit from it.
You should not create a data model that includes a complex type. You would create a data model with a complex type if you wanted an entity property in the data model to also have properties.
You should not create a data model that implements a single entity with multiple associations. To implement single-table inheritance in this scenario, you must create a base entity and entities that inherit from the base entity. A single entity would not suffice. Associations are used to create logical relationships between entities in an EDM.

QUESTION 5
You need to configure a security solution for an application. The solution must meet the following requirements:
·
The application must have access to tables in a database ·
The tables must only be accessed through the application ·
Database access must not require a password
What should you create?

A. a database user that has no login
B. a new login that has a blank password

C. an application role
D. a proxy object

Answer: A
Section: (none)

Explanation/Reference:

QUESTION 6
You are a developer on an instance of SQL Server 2008. You are designing an **Event** table. The **EventDate** column must contain a value. In addition, for each event, the **EventDate** must be a date between the **StartDate** and **EndDate**.
Which two constructs should you use? (Choose two. Each correct answer represents a portion of the answer.)

A. a `DEFAULT` definition
B. a `NOT NULL` constraint
C. a table-level `CHECK` constraint
D. a `UNIQUE` constraint
E. a column-level `CHECK` constraint

Answer: BC
Section: (none)

Explanation/Reference:
A `NOT NULL` constraint is used to ensure that a non-null value is specified for a column when a new row is added to a table or when a row in a table is updated. To ensure that a non-null value will be specified for the **EventDate** column for a new row in the **Event** table, you should specify a `NOT NULL` constraint on the **EventDate** column. If an attempt is made to insert a row into the **Event** table without specifying an event date, the insert will fail and an error message will be returned. To ensure the event date is within the desired date range, you can use a table-level `CHECK` constraint. A `CHECK` constraint is used to restrict the data that is allowed for a column to specific values. A `CHECK` constraint consists of a Boolean expression that evaluates to either `TRUE` or `FALSE`. If the expression evaluates to `TRUE`, the value is allowed for the column, and if the expression evaluates to `FALSE`, the value is not allowed for the column. `CHECK` constraints can be defined at the table level or column level, but only `CHECK` constraints defined at the table level can use columns other than the constrained column in the constraint expression. In this scenario, you need to create a `CHECK` constraint on the **EventDate** column and use other table columns in the constraint expression. Therefore, you must use a table-level constraint.
You should not use a `DEFAULT` definition. A `DEFAULT` definition is used to specify a value that should be assigned to a column when a user adds a row without specifying a value for that column. A `DEFAULT` definition can include constants and operators, but cannot contain column names.
You should not use a column-level `CHECK` constraint. In this scenario, the constraint must reference the **EventDate**, **StartDate**, and **EndDate** columns. To create a constraint that accesses columns other than the constrained column, you must create a table-level `CHECK` constraint.
You should not use a `UNIQUE` constraint. A `UNIQUE` constraint is used to ensure that a column contains unique, non-duplicate values. Unlike a `PRIMARY KEY` constraint, which also ensures non-duplicate values, a `UNIQUE` constraint allows a single row to have a `NULL` value for the column. When you define a column with a `UNIQUE` constraint, SQL Server automatically creates a unique index on the column and uses this index to prevent duplicate values.

QUESTION 7
You are a database administrator on an instance of SQL Server 2008. You need to perform complex

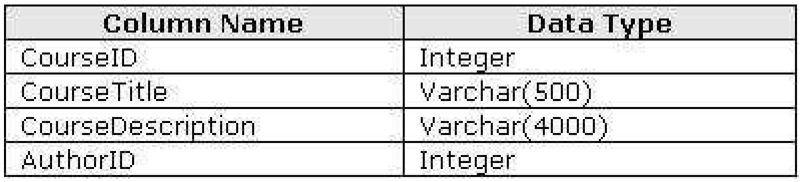
processing on data from the **OrderHistory**
table in your database. You need the ability to scroll backward and forward through the rows in the table and to perform specific actions on rows.
You always need to have access to the most up-to-date data.
Which construct should you use?

A. a dynamic cursor
B. a `SELECT...INTO` statement
C. a `SELECT` statement that includes the `OUTPUT` clause
D. a partitioned view

Answer: A
Section: (none)

Explanation/Reference:
You should use a dynamic cursor. Cursors can be used to operate on underlying data on a row-by-row basis. A dynamic cursor allows scrolling
forward and backward, and all data changes to the underlying table are visible. Cursors are sometimes unavoidable, but should be replaced with
set-based operations or other methods when possible because they can often degrade performance.
You should not use a `SELECT...INTO` statement. The `SELECT...INTO` statement selects data from a table and inserts the data into a newly
created table. This could not be used to implement the required functionality.
You should not use a `SELECT` statement that includes the `OUTPUT` clause. When performing DML operations, you can use the `OUTPUT` clause to
obtain and display information about the rows affected by the DML operation. The `OUTPUT` clause can display this information to the user, insert the
data into another permanent or temporary table or table variable using an `INTO` clause, or pass the data to a nested DML statement for processing.
You should not use a partitioned view. Partitioned views are used when you have similar data stored in multiple tables and want to create a view to
allow access to all of the data as if it were stored in a single table. Partitioned views are implemented by creating a view that queries several tables
and combines the results using the `UNION ALL` operator. A partitioned view can improve performance and increase availability, but would not be
applicable in this scenario.

QUESTION 8
You are a database developer. You plan to design a database solution by using SQL Server 2008. You have a Web site supported by a database that has the full-text search component installed. You plan to create a table named Courses that will have the following structure.



Column Name	Data Type
CourseID	Integer
CourseTitle	Varchar(500)
CourseDescription	Varchar(4000)
AuthorID	Integer

Users of the Web site will search for courses based on the CourseTitle field. You need to construct a full-text query that ensures the following compliances when a user launches the search for a course:
·
Rows are returned when the exact search phrase is found.
·
Rows are in order of how well they match with the search phrase.

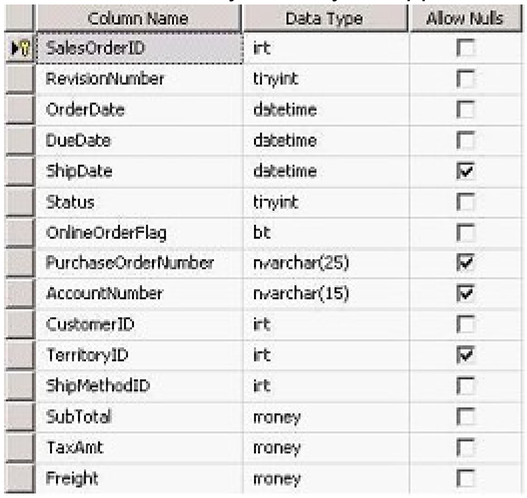
What should you specify in the full-text query?

A. A FREETEXT predicate
B. A CONTAINS predicate
C. A FREETEXTTABLE function
D. A CONTAINSTABLE function

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 9
Your **Prod** database resides on an instance of SQL Server 2008. The **Prod** database contains a **SalesHeader** table that is used by one of your applications. The **SalesHeader** table is defined as follows:



Column Name	Data Type	Allow Nulls
SalesOrderID	int	☐
RevisionNumber	tinyint	☐
OrderDate	datetime	☐
DueDate	datetime	☐
ShipDate	datetime	✓
Status	tinyint	☐
OnlineOrderFlag	bit	☐
PurchaseOrderNumber	nvarchar(25)	✓
AccountNumber	nvarchar(15)	✓
CustomerID	int	☐
TerritoryID	int	✓
ShipMethodID	int	☐
SubTotal	money	☐
TaxAmt	money	☐
Freight	money	☐

Currently, the **SalesHeader** table has a clustered index on the **SalesOrderID** column, and no other indexes.
You want to create indexes to provide best performance to your application.
Your application allows users to perform the following actions:
Searching for an order with a specified order number
Searching for orders within a specified order number range
Searching for orders that belong to specific customers
Searching for orders with a specified customer account number

The application will use the **SalesOrderID** and **CustomerID** columns in joins to retrieve data from other tables. Which index or indexes should you create?

A. a full-table nonclustered index on the **CustomerID** column and a filtered index on the **AccountNumber** column
B. a clustered index on the **CustomerID** column
C. a nonclustered index on the **CustomerID** column and a nonclustered index on the **AccountNumber** column
D. a composite non-clustered index on the **CustomerID** and **AccountNumber** columns

Answer: C
Section: (none)

Explanation/Reference:

In this scenario, the application needs to search based on the order number and a range of ordered numbers. Searches on the order number would already be optimized because a clustered index already exists on the **SalesOrderID** column. To support searches by customer, you should create a nonclustered index on the **CustomerID** column, and to support searches by account number, you should create a nonclustered index on the **AccountNumber** column.
You cannot create a clustered index on the **CustomerID** column because the table already contains a clustered index on the **SalesOrderID** column. A table can only have one clustered index, because the index represents the physical order of the rows in the table.
You should not create a composite non-clustered index on the **CustomerID** and **AccountNumber** columns. A composite index is helpful when searching for both columns, but in this scenario, the application does not allow searching by both customer ID and account number.
You should not create a full-table nonclustered index on the **CustomerID** column and a filtered index on the **AccountNumber** column. Although a nonclustered index on the **CustomerID** column would improve searches for specific customers, there is no need to create a filtered index on the **AccountNumber** column. A filtered index is a nonclustered index that is defined including a `WHERE` clause to optimize the index for queries that access specific subsets of the data.

QUESTION 10
You are a database developer on an instance of SQL Server 2008. You want to perform full-text searches using the **Details** column of the **Product** table, but you want these searches to bypass certain words.
Which full-text component should you use to accomplish this?

A. a full-text search filter
B. a stoplist
C. a thesaurus
D. a dictionary

Answer: B
Section: (none)

Explanation/Reference:
. A stoplist contains a list of words that are ignored in full-text searches. These words are also known as noise words. When a full-text index is created, you can include the `WITH STOPLIST` clause to specify the stoplist that should be used for the full-text index. These stopwords are not included in the full-text index. By default, SQL Server uses a system stoplist that contains meaningless words like articles and prepositions. However, you can create your own stoplist and add words to it. For example, the following statements create a new stoplist, named **CustomStoplist**, from the system stoplist and add the word unknown to the stoplist:
```
CREATE FULLTEXT STOPLIST CustomStoplist FROM SYSTEM STOPLIST;ALTER FULLTEXT
STOPLIST CustomStoplist ADD 'unknown';
```
After creating the stoplist, you specify the stoplist when creating the full-text index as follows:
```
CREATE FULLTEXT INDEX ON dbo.Product(Details) KEY INDEX PK_Product_ProductID WITH
STOPLIST = CustomStoplist;
```
Previous versions of SQL Server used noise-word files instead of stoplists. Any modifications to the default noise-word file will not be included in the system stoplist when the database is upgraded to SQL Server 2008. You must manually modify the system stoplist after upgrading to SQL Server 2008.
You should not use a full-text search filter. Full-text search filters are used when performing full-text searches on **varbinary**, **varbinary(max)**, **image**, and **xml** data type columns. The filter processes the document to obtain the text information that should be used in the search.
You should not use a thesaurus. Thesaurus files identify words with similar meanings. Thesaurus files are used with queries that use the `FREETEXT` predicate and the `FREETEXTTABLE` function, and with queries that use the `CONTAINS` predicate and the `CONTAINSTABLE` function with the `FORMS OF THESAURUS` clause. A global thesaurus file, named **tsGlobal.xml**, and a thesaurus file for each language are located in the **SQL_Server_install_path\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\FTDATA\** folder. These default thesaurus files have XML elements commented out initially, but you can modify them as required to add or remove synonyms.

You should not use a dictionary. A dictionary is not a valid full-text index component. SQL Server does use a thesaurus file to identify synonyms, but does not use a dictionary.

QUESTION 11
You are designing a table that will store transactions for an online retailer. You anticipate that the online retailer will have an average of 500,000 transactions per day. You need to design a solution for data aging and archiving. The solution must minimize the amount of time it takes to return query results for active data.
What should you implement?

A. a linked server
B. a table schema
C. Service Broker
D. table partitioning

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 12
You are a database developer on an instance of SQL Server 2008. Your **Prod** database is accessed extensively by application users and users performing ad hoc queries.
Using SQL Profiler, you have identified several queries that are performing poorly. You are reviewing their execution plans to determine possible ways of optimizing them.

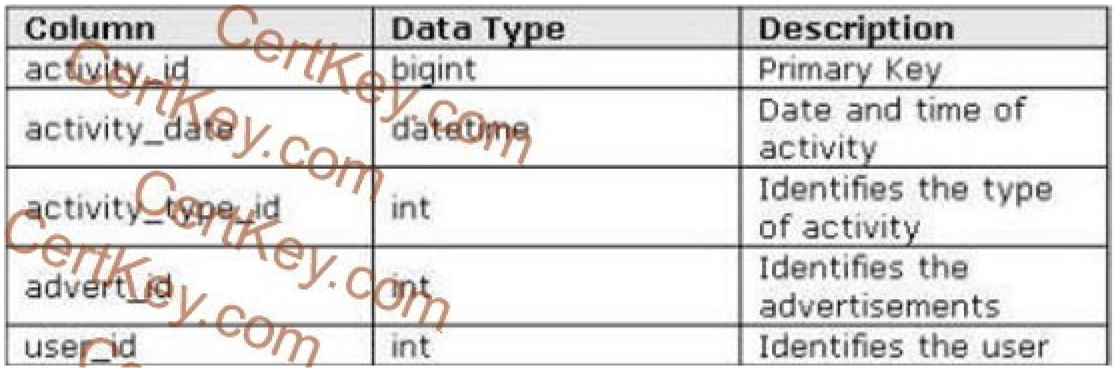
Which operator in an execution plan would more likely have an adverse effect on memory requirements for a query performance?

A. a **Nonclustered Index Seek** operator
B. a **Hash Match** operator
C. a **Nested Loop** operator
D. a **Merge Join** operator

Answer: B
Section: (none)

Explanation/Reference:
A **Hash Match** operator would have an adverse effect on memory requirements for a query. Hash joins read rows in a table, hash the join column, and store the results in a hash table to use with the remaining query input. Hash joins are more memory-intensive, and can cause the **tempdb** database to grow significantly.
All of the other options are incorrect because they are less likely to adversely affect memory requirements for a query.
A **Nested Loop** operator is used to perform some types of joins. It iterates through all the rows of one input and, for each row in the input, iterates through all the rows of another input to search for matches. This operation is preferred when one or both the tables have only a few rows. If either table has a large number of rows, this operation may be expensive, because the second input is read once for each row in the first input.
A **Nonclustered Index Seek** operator performs a seek operation on a nonclustered index. This operator is usually preferred over a **Nonclustered Index Scan**.
A **Merge Join** operator is used to perform some types of joins. It reads through both of the inputs one time, performing comparisons to determine which rows to return. This operation requires that both inputs be sorted on the join columns.

QUESTION 13
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. The database captures user interactions and supports a Web site. The Activity table of the User_Activity database stores these interactions. Data older than half a year is archived to the Activity table of the Archive_Activity database. This Archive_Activity database is on a different instance of SQL Server 2008. The table below shows the structure of the Activity table. You intend to design a solution design a solution. The solution allows a single query to generate a report. User interactions for the last 12 months are summarized by the report. So what should you do to make sure that the solution is implemented?



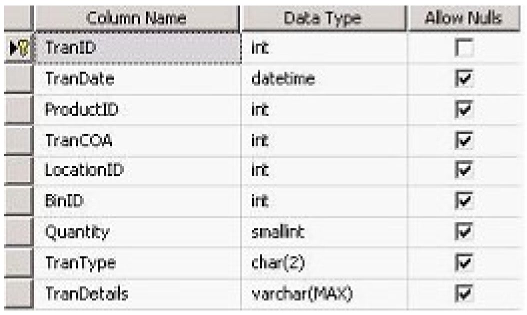
Column	Data Type	Description
activity_id	bigint	Primary Key
activity_date	datetime	Date and time of activity
activity_type_id	int	Identifies the type of activity
advert_id	int	Identifies the advertisements
user_id	int	Identifies the user

A. A partition function and a partition scheme should be created
B. The archived data should be moved to the User_Activity database
C. The Activity tables should be modified to use the partition scheme
D. You should use the UNION ALL clause to create a view. Use a view to retrieve data from the two Activity tables.
E. on the two Activity tables, CHECK constraints should be created to limit the values in the activity_date column to an exclusive range.

Answer: DE
Section: (none)

Explanation/Reference:

QUESTION 14
You are a database developer on an instance of SQL Server 2008. You have a **Prod** database that contains all inventory-related data for your company.
Your **InvTransaction** table is defined as follows:



Column Name	Data Type	Allow Nulls
TranID	int	☐
TranDate	datetime	✓
ProductID	int	✓
TranCOA	int	✓
LocationID	int	✓
BinID	int	✓
Quantity	smallint	✓
TranType	char(2)	✓
TranDetails	varchar(MAX)	✓

You have a Transact-SQL script that is performing poorly, and you need to determine the cause of the degraded performance. The script executes an initial `SELECT` query on the **InvTransaction** table and then uses the retrieved data values as criteria in additional queries. You view the graphical execution plan. Which indicator would you most likely investigate further?

A. a query that has a query cost value of 0
B. an operator that indicates a full table scan
C. an operator with a low cost percentage
D. a sort operator

Answer: B
Section: (none)

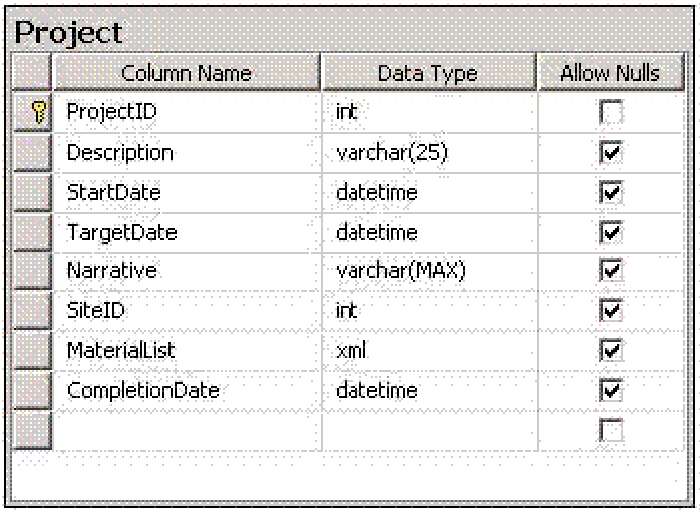
Explanation/Reference:
Although full table scans on small tables are not significant, full table scans on extremely large tables can be expensive. You can usually improve performance by creating indexes to be used instead of scanning the entire table.
A query that has a query cost value of 0 would not be an issue to investigate further. Each query in a batch of Transact-SQL is assigned a query cost value based on its cost relative to the entire batch. In a batch that includes only one statement, the query cost value is 100. You would choose to investigate queries within a multi-statement batch if they had an extremely high query cost value relative to the batch.
An operator with a low cost percentage would not be an issue you would investigate further. A low cost percentage for an operator indicates that the operation is performed without much performance impact. You would investigate operators that had a high cost percentage.
A sort operator alone would not be cause for further investigation. However, you might want to investigate a sort operator with a high cost percentage. Sorts can consume resources, so you should carefully evaluate how your query performs sort operations. You might choose to eliminate `ORDER BY` clauses that are not necessary, or choose to create indexes using a specific order.

QUESTION 15
You are a database developer on an instance of SQL Server 2008. Your **Project** table is defined as follows:
(Click on exhibit to view table structure)
You need to construct an XML document to send to the corporate office. The XML document needs to contain data from non-**xml** columns in the table, as well as from the **MaterialList** column, which contains XML in the following format:
```
<Materials>
 <Item>
  <SKU ID="XK-33" ItemDesc="Corner Brackets" />
  <Quantity>15</Quantity>
  <RequiredDt>03-15-2009</RequiredDt>
 </Item>
 <Item>
  <SKU ID="XV-17" ItemDesc="Metal Shelving" />
  <Quantity>100</Quantity>
  <RequiredDt>03-12-2009</RequiredDt>
 </Item>
</Materials>
```

You want to create a stored procedure to extract non-**xml** columns in the **Project** table as an **xml** data type, assign it to an **xml** variable, and perform additional XQuery processing to construct the output XML document.

Which query should you use within the stored procedure to extract the non-**xml** columns?

Exhibit:



Project

	Column Name	Data Type	Allow Nulls
🔑	ProjectID	int	☐
	Description	varchar(25)	✓
	StartDate	datetime	✓
	TargetDate	datetime	✓
	Narrative	varchar(MAX)	✓
	SiteID	int	✓
	MaterialList	xml	✓
	CompletionDate	datetime	✓
			☐

A. a query that uses `OPENXML`
B. a query with that uses the **query()** method with an XQuery expression
C. a query with `FOR XML` that includes the `TYPE` directive
D. a query that uses a FLWOR expression with a **where** clause

Answer: C
Section: (none)

Explanation/Reference:
The `FOR XML` clause specifies that the result of the `SELECT` statement should be returned in XML format. Including the `TYPE` directive will ensure that the result is returned as typed XML instead of in text form. This will allow you to assign the result to an **xml** variable and process it as required. The `TYPE` directive should be used when you need the results of a query using `FOR XML` returned as an XML instance, as when you need to use a query containing `FOR XML` as a subquery of another query that uses `FOR XML`.
You should not use a query that uses `OPENXML`. `OPENXML` is used to create a relational view of XML data, and in this scenario, you were retrieving relational data as XML. The `OPENXML` function can be used in `SELECT` statements where a table or view would be specified to extract data from an XML document. The `WITH` clause can be used to specify a table name that defines the relational structure of the result.
You should not use a query with that uses the **query()** method with an XQuery expression. The **query()** method is used to query and retrieve XML elements and attributes from an XML instance, not to retrieve data from non-**xml** columns. The method accepts a string XQuery expression that determines the elements and element attributes that are extracted, and returns untyped XML.
You should not use a query that uses a FLWOR expression with a **where** clause. FLWOR expressions allow you to use XQuery to process XML using a construct similar to a SQL `SELECT` statement. The term FLWOR is derived from the clauses used in the expression, namely **for**, **let**, **where**, **order by**, and **return**. The clauses of a FLWOR expression are defined as follows:
for: Defines the elements processed by the expression. This required clause is similar to the `FROM` clause of a `SELECT` statement.
let: Defines a variable that can be used in the expression. This optional clause can include calls to XQuery functions. This clause was not allowed in previous SQL Server versions, but is supported in SQL Server 2008.

where: Filters the result returned. This optional clause is similar to the `WHERE` clause of a `SELECT` statement.
• **order by**: Orders the result returned. This optional clause is similar to the `ORDER BY` clause of a `SELECT` statement.
• **return**: Returns the result. This clause specifies the XML nodes that will be returned.
Variables in a FLWOR expression are prefixed with a dollar sign (`$`). The FLWOR expression iterates over the

XML nodes specified by the **for** clause and returns the specified result.

QUESTION 16

You are a database developer on an instance of SQL Server 2008. You created the **TestDetails** table using the following `CREATE TABLE` statement:

```
CREATE TABLE TestDetails (TestID int IDENTITY(1,1) PRIMARY KEY, TestName nvarchar
(10) NULL UNIQUE,TestType char(1) CHECK (TestType = 'A' OR TestType = 'B'),
TestAuthorID int SPARSE);
```

With default settings, which row would be successfully inserted into the **TestDetails** table?

A. a row that has an explicit value for the **TestID**

B. a row that has a `NULL` value for **TestID**

C. a row that has a non-`NULL` value for **TestAuthorID**

D. a row that has a value of 'C' for **TestType**

Answer: C
Section: (none)

Explanation/Reference:
The **TestAuthorID** is defined as a sparse column. When creating a table, sparse columns can be used to optimize the storage of `NULL` values; however, they do not affect the nullability of a column. With the given table, you could insert a `NULL` value or a non-`NULL` value for the **TestAuthorID** column. You can use `NOT NULL` in a column's definition to disallow `NULL` values for a column. However, sparse columns cannot be defined with a `NOT NULL` constraint.
A row that has an explicit value for **TestID** would not be successfully inserted into the **TestDetails** table. By default, identity columns are assigned the next identity value when a record is inserted. Attempting to include an `IDENTITY` column in the column list and `VALUES` clause of an `INSERT` statement will result in the following error:

```
Msg 544, Level 16, State 1, Line 14Cannot insert explicit value for identity
column in table 'TestDetails' when IDENTITY_INSERT is set to OFF.
```

However, you can override this behavior and allow an `IDENTITY` column to be explicitly set. The following statement specifies that the **TestDetails** table should accept an explicit **TestID** value without generating an error:

```
SET IDENTITY_INSERT TestDetails ON;
```

A row that has a `NULL` value for **TestID** would not be successfully inserted into the **TestDetails** table because the **TestID** column is defined as the table's primary key. A primary key must contain a unique, non-`NULL` value. If you attempted to insert a `NULL` value for the **TestID**, the statement would generate the following error:

```
Msg 339, Level 16, State 1, Line 14DEFAULT or NULL are not allowed as explicit
identity values.
```

A row that has a value of 'C' for **TestType** would not be successfully inserted into the **TestDetails** table because the **TestType** column has a `CHECK` constraint. `CHECK` constraints are enabled by default, and attempting to add a row that violates the `CHECK` constraint would generate an error similar to the following:

```
Msg 547, Level 16, State 0, Line 14The INSERT statement conflicted with the CHECK
constraint "CK__TestDetai__TestT__6CA31EA0". The conflict occurred in database
"KIT3", table "dbo.TestDetails", column 'TestType'.
```

QUESTION 17

You intend to use SQL Server 2008 to create a database solution.

There's a large table in the database.
The table is updated not very often. A query is executed against the table by users.
The execution of a complex calculation is required by the query.
The calculation involves multiple columns for a given row.

You notice that because the query is CPU intensive, the query has a bad performance.

So what should you do to minimize the effect of this query on the server?

A. A view should be created on the table.
B. An index should be created on each field.
C. On the table, a computed column should be created.
D. On the table, a persisted computed column should be created

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 18
You have to design a new database, the tables in the database should be replicated to three offices and the database tables should have the following requirements :

- a unique row should be identified among all the rows
- the row should be unique among the entire organization.

What option should you use?

A. add a new column that have the identity property.
B. add a new column that have the hierarchyid datatype.
C. add a new column with a uniqueidentifier datatype and put the default to the getdate().
D. add a new column with a uniqueidentifier datatype and put the default to the NEWID().

Answer: D
Section: (none)

Explanation/Reference:
http://msdn.microsoft.com/en-us/library/ms190348.aspx

QUESTION 19
You are a database developer. You plan to create a database by using SQL Server 2008.
The database will store information about students, teachers, classes, and rooms in a school.
The database will be used by a scheduling application.

In the design plan, the following facts have to be considered:
- Each teacher can teach one or more classes.
- Each student can register for one or more classes.
- Each class can be in one or more rooms.
- Each room can host one or more classes.

You identify the following entities for the database design:
- Students
- Teachers
- Classes
- Rooms
- ClassesStudents
- ClassesTeachers

You need to design the database to ensure normalization.What should you do?

A. 1. Add a new entity named TeachersStudents.2. Establish a relationship between the Teachers and Students entities by using the TeachersStudentsentity.
B. 1. Add a new entity named ClassesRooms.2. Establish a relationship between the Classes and Rooms entities by using the ClassesRooms entity.
C. 1. Add a new entity named TeachersRooms.2. Establish a relationship between the Teachers and Rooms entities by using the TeachersRoomsentity
D. 1. Create a new entity named StudentsRooms.2. Establish a relationship between the Students and Rooms entities by using the StudentsRooms entity.

Answer: B
Section: (none)

Explanation/Reference:

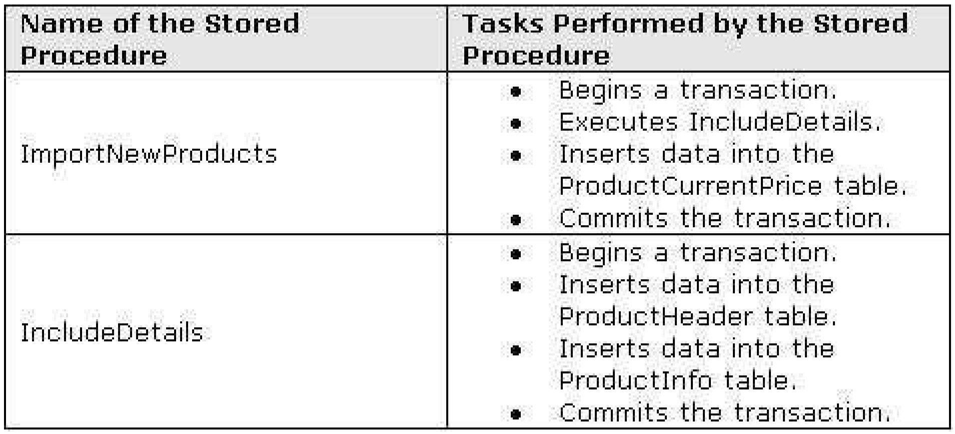
QUESTION 20
You are designing a database for a reporting solution that is based on data from an Online Transaction Processing (OLTP) database. The reports will contain aggregated data. You need to ensure that the reports will not affect query performance on the OLTP database. The solution must minimize the use of joins when performing the aggregate calculations.
What should you do?

A. Add a persisted computed column.
B. Create indexed views in the OLTP database.
C. Create partitioned tables in the OLTP database.
D. Create a new denormalized database based on the OLTP database.

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 21
You are a database developer. You develop solutions by using SQL Server 2008 in an enterprise environment. An application contains two stored procedures. The tasks performed by the stored procedures are as shown in the following table.



Name of the Stored Procedure	Tasks Performed by the Stored Procedure
ImportNewProducts	• Begins a transaction. • Executes IncludeDetails. • Inserts data into the ProductCurrentPrice table. • Commits the transaction.
IncludeDetails	• Begins a transaction. • Inserts data into the ProductHeader table. • Inserts data into the ProductInfo table. • Commits the transaction.

You discover that the procedures occasionally throw foreign key violation errors. IncludeDetails throws an error when it inserts records into the ProductInfo table. ImportNewProducts throws an error when it inserts records into the ProductCurrentPrice table.

.
If an error occurs in the INSERT statement of ProductInfo, records inserted into ProductHeader www.
Dump4certs.com
and ProductCurrentPrice are committed.
.
If an error occurs in the INSERT statement of ProductCurrentPrice, all transactions are rolled back.
What should you do?

A. 1. Add a SET XACT_ABORT OFF statement in IncludeDetails.
   2. Add a SET XACT_ABORT ON statement in ImportNewProducts.
B. 1. Add a SET XACT_ABORT ON statement in IncludeDetails.
   2. Add a SET XACT_ABORT OFF statement in ImportNewProducts.
C. 1. Enclose all statements of IncludeDetails in a TRY/CATCH block.
   2. Add a ROLLBACK TRANSACTION statement in the CATCH block.
D. 1. Enclose all statements of ImportNewProducts in a TRY/CATCH block.
   2. Add a ROLLBACK TRANSACTION statement in the CATCH block.

Answer: A
Section: (none)

Explanation/Reference:

QUESTION 22
You are the database developer on an instance of SQL Server 2008. You create a table named **Realtors** using the following statement:
```
CREATE TABLE Realtors (RealtorID int PRIMARY KEY,LastName varchar(35) NOT NULL,
FirstName varchar(25) NOT NULL,LicenseNo int NOT NULL);
```
The **LicenseNo** column will contain a numeric license number that is assigned by an external licensing board.
You need to ensure that no realtors in the **Realtors** table have the same license number.
What should you do?

A. Add a `UNIQUE` constraint on the **LicenseNo** column.
B. Create a table-level `CHECK` constraint on the **LicenseNo** column.
C. Create a trigger on the **Realtors** table to check for duplicate license numbers.

D. Add a `DEFAULT` definition on the **LicenseNo** column.

Answer: A
Section: (none)

Explanation/Reference:
This will prevent duplicate values from being inserted into the **LicenseNo** column of the table. When a `UNIQUE` constraint is defined, a unique index is automatically created to enforce the constraint. In this scenario, you could use the following statement to add a `UNIQUE` constraint to the **LicenseNo** column of the **Realtors** table:

```
ALTER TABLE dbo.Realtors ADD CONSTRAINT uqc_Realtors_LicenseNo UNIQUE
(LicenseNo);
```

After executing this statement, each row in the **Realtors** table must have a non-null, unique **LicenseNo** value. If a user attempted to insert a row into the **Realtors** table containing a duplicate **LicenseNo** value, an error message similar to the following would be displayed:

```
Msg 2627, Level 14, State 1, Line 5Violation of UNIQUE KEY constraint
'uqc_Realtors_LicenseNo'. Cannot insert duplicate key in object 'dbo.Realtors'.
```

`PRIMARY KEY` constraints can also enforce uniqueness, but do not allow the key column(s) to contain a `NULL` value. A `UNIQUE` constraint will allow a single `NULL` value. However, in this scenario, the **Realtors** table already contains a primary key column, **RealtorID**. Therefore, an additional `PRIMARY KEY` constraint could not be used.
You should not create a table-level `CHECK` constraint on the **LicenseNo** column. A `CHECK` constraint restricts the data that is allowed for a column to specific values. A `CHECK` constraint consists of a Boolean expression that evaluates to either `TRUE` or `FALSE`. If the expression evaluates to `TRUE`, the value is allowed for the column, and if the expression evaluates to `FALSE`, the value is not allowed for the column. In addition, `CHECK` constraints defined on a column can only reference the given column, not other columns in the table. To reference other columns within the table, the `CHECK` constraint must be defined at the table level rather than the column level.
You should not create a trigger on the **Realtors** table to check for duplicate license numbers. Before implementing a trigger, you should consider whether the same result could be achieved using a constraint. In this scenario, a `UNIQUE` constraint would perform the same tasks that a trigger could perform. Constraints should be used instead of triggers when possible because triggers are more complex to create and maintain, and usually require more overhead.
You should not add a `DEFAULT` definition on the **LicenseNo** column. A `DEFAULT` definition is used to specify a value that should be assigned to a column if the user does not explicitly specify a value for the column when adding a row. A `DEFAULT` definition can include constants and operators, but cannot contain column names.

QUESTION 23
You need to provide a developer the ability to create and modify database diagrams from an existing database by using SQL Server Management Studio. The solution must minimize the amount of permissions assigned to the developer.

What should you do?

A. Add the developer to the sysadmin role.
B. Add the developer to the db_owner role.
C. Grant the developer the CREATE TABLE permission only.
D. Grant the developer the CREATE SCHEMA permission only.

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 24
You plan to deploy a new application. The application will perform the following operations:
Create a new database
Add new logins
Back up the new database

You need to configure a login to support the deployment of the new application. The solution must ensure that the application uses the most restrictive permissions possible.
What should you do?

A. Add the login to the sysadmin server role.
B. Add the login to the dbcreator and securityadmin server roles.
C. Add the login to the diskadmin and securityadmin server roles.
Once the database is created, add a user to the db_backupoperator database role.
D. Add the login to the diskadmin and serveradmin server roles.
Once the database is created, add a user to the db_backupoperator database role.

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 25
You are a database developer on an instance of SQL Server 2008. You are developing a Transact-SQL script that will be used by developers. The
script uses multiple DML statements, but does not use transactions. You want to ensure that each DML statement is treated as a separate transaction, but not automatically committed when executed. You must also ensure that all DML statements are explicitly committed or all are rolled back.
You want to accomplish this with minimum effort. What should you do?

A. Add the `SET IMPLICIT_TRANSACTIONS ON` statement.
B. Add the `SET IMPLICIT_TRANSACTIONS OFF` statement.
C. Set the transaction isolation level to `READ COMMITTED`.
D. Include a custom error-handler for each DML statement.

Answer: B
Section: (none)

Explanation/Reference:
The `IMPLICIT_TRANSACTIONS` option controls how transactions are handled. Transactions can be specified explicitly, implicitly, or automatically. In autocommit mode, which is the default, each Transact-SQL statement is treated as a separate transaction, and each statement is automatically committed when it successfully executes. In this scenario if you set the `IMPLICIT_TRANSACTIONS` option to `ON`, transactions will be implicitly created. This means that transactions will be automatically started if there is no current transaction, and must be explicitly committed or rolled back. Any uncommitted implicit transactions are rolled back when the user disconnects.
You should not add the `SET IMPLICIT_TRANSACTIONS OFF` statement. This is the default setting, and all Transact-SQL would use autocommit mode. Each DML statement would be considered a separate transaction and automatically committed if it executed successfully.
You should not set the transaction isolation level to `READ COMMITTED`. The transaction isolation level controls how a transaction behaves when there are other concurrent transactions. However, in this scenario,

transactions are not used. Therefore, this would not be applicable.
You should not include a custom error-handler for each DML statement because this would require more effort than necessary. You can check the value returned by the `@@ERROR` function for each DML statement and include blocks of code to perform the desired actions. However, in this scenario, you want to have minimal development effort.

QUESTION 26
You are a database developer. You develop a task management application that connects to a SQL Server 2008 database named TaskDB.
Users log on to the application by using a SQL Server login.
The application contains a module named Task that assigns tasks to users.
Information about these tasks is stored in the Tasks table of the TaskDB database.
The Tasks table contains multiple columns.
These include the CloseDate and EstimatedTime columns.
Users assigned to a database role named User1 can update all task information columns except the CloseDate and the EstimatedTime columns in the Tasks table.
Administrative users assigned to a database role named Task_Admin can update all task information in the Tasks table.

You need to design a strategy to meet the security requirements. Which two actions should you perform? (Each correct answer presents part of the solution. Choose two.)

A. Add the Task_Admin role to the db_accessadmin fixed database role.
B. Grant Update permissions on the Tasks table to the Task_Admin role.
C. Grant Update permissions on the Tasks table to the User1 role for each column except the CloseDateand EstimatedTime columns.
D. Create an INSTEAD OF trigger on the Tasks Table.Use the Is_Member function to prevent the User1 role from updating the CloseDate and EstimatedTimecolumns.

Answer: BC
Section: (none)

Explanation/Reference:

QUESTION 27
You are a database developer. You plan to design a database solution by using SQL Server 2008. A database contains a table that has a column defined as a smallint data type. The table is partitioned on has boundaries of 100 and 1,000.
The table must be altered to contain the following partitions:
< 100
>= 100 and < 400
>= 400 and < 700
>= 700 and < 1000
>= 1000
You need to alter the partition function to provide the required partitions.
Which code fragment should you use?

A. ALTER PARTITION FUNCTION MyRangePF1 () SPLIT RANGE (399); GO ALTER PARTITION FUNCTION MyRangePF1 () SPLIT RANGE (699); GO
B. ALTER PARTITION FUNCTION MyRangePF1 () SPLIT RANGE (400); GO ALTER PARTITION FUNCTION MyRangePF1 () SPLIT RANGE (700); GO

C. DROP PARTITION FUNCTION myRangePF1; GO
   CREATE PARTITION FUNCTION myRangePF1 (smallint)
   AS RANGE RIGHT FOR VALUES (99, 399, 699, 999);

D. DROP PARTITION FUNCTION myRangePF1; GO
   CREATE PARTITION FUNCTION myRangePF1 (smallint)
   AS RANGE LEFT FOR VALUES (100, 400, 700, 1000);

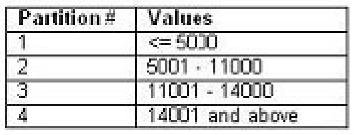
Answer: B
Section: (none)

Explanation/Reference:

QUESTION 28
You are a database developer on an instance of SQL Server 2008. You create a partition function, partition scheme, and partitioned table using the following Transact-SQL statements:

```
CREATE PARTITION FUNCTION Pf1 (int)AS RANGE LEFT FOR VALUES (5000, 8000, 11000,
14000);
CREATE PARTITION SCHEME Ps1 AS PARTITION Pf1 TO (fg1, fg2, fg3, fg4, fg5);
CREATE TABLE PTable1 (ID int,Description nvarchar(40),Code int)ON Ps1 (Code);
```

After creating **PTable1** and inserting data into the table, you decide that the table should only contain four partitions, defined as follows:



Partition #	Values
1	<= 5000
2	5001 - 11000
3	11001 - 14000
4	14001 and above

Which Transact-SQL statement should you execute?

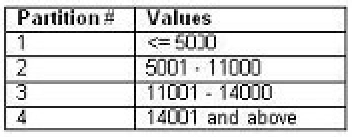
A.
```
ALTER PARTITION FUNCTION Pf1()
SPLIT RANGE (8000, 11000);
```
B.
```
ALTER PARTITION FUNCTION Pf1()
MERGE RANGE (8000);
```
C.
```
ALTER PARTITION FUNCTION Pf1()
MERGE RANGE (8001);
```
D.
```
ALTER PARTITION FUNCTION Pf1()
SPLIT RANGE (8000);
```

Answer: B
Section: (none)

Explanation/Reference:
In this scenario, you originally created a partition function using the `CREATE PARTITION FUNCTION` statement. A partition function maps the rows of a table or index into partitions based on the specified partition boundary values. The complete syntax for the `CREATE PARTITION FUNCTION` statement is:

```
CREATE PARTITION FUNCTION name_of_partition_function (type_of_input_parameter)AS
RANGE [LEFT | RIGHT]FOR VALUES ([boundary_value [,...n]] );
```

The `CREATE PARTITION FUNCTION` statement in this scenario created a partition function named **Pf1** with a partition column of the **int** data type. The `FOR VALUES` clause of the `CREATE PARTITION FUNCTION` statement specifies the boundary value of each partition. The `RANGE RIGHT` and `RANGE LEFT` clauses are used to specify how the actual boundary values are handled. `RANGE LEFT` indicates that the boundary value should be stored in the partition on the left side of the boundary value, with the boundary values sorted in ascending order from left to right. The `CREATE PARTITION FUNCTION` statement in this scenario defined five partitions as follows:

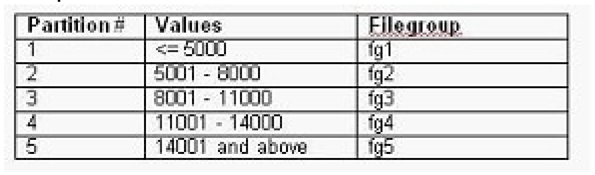


Partition #	Values
1	<= 5000
2	5001 - 11000
3	11001 - 14000
4	14001 and above

Next, you create a partition scheme using the `CREATE PARTITION SCHEME` statement. The `CREATE PARTITION SCHEME` statement is used to create a partition scheme. A partition scheme maps the different partitions created by the partition function to filegroups. The `AS PARTITION` clause of the `CREATE PARTITION SCHEME` statement identifies the partition function, and the `TO` clause specifies the filegroups. The complete syntax for the `CREATE PARTITION SCHEME` statement is:

```
CREATE PARTITION SCHEME name_of_partition_scheme AS PARTITION
name_of_partition_function [ALL] TO ({ file_group | [PRIMARY]} [,...n] );
```

The partition scheme created in this scenario would map the partitions to file groups as follows:



Partition #	Values	Filegroup
1	<= 5000	fg1
2	5001 - 8000	fg2
3	8001 - 11000	fg3
4	11001 - 14000	fg4
5	14001 and above	fg5

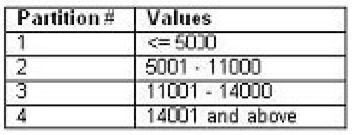
Finally, you create a partitioned table using the `CREATE TABLE` statement that includes an `ON` clause. The `ON` clause identifies the partition scheme
and the column in the table on which partitioning will be performed. The specified partitioning scheme identifies the partition function that is used. The complete syntax of the `CREATE TABLE` statement to create a partitioned table is as follows:

```
CREATE TABLE table_name (column_def1, column_def2, ...)ON
name_of_partition_scheme (partition_column_name);
```

After creating the table and inserting data, you wanted to modify the partitioning of the table to combine partition #2 and partition #3 into a single partition. You can use the `ALTER PARTITION FUNCTION` statement to add a partition to an existing partitioned table, or to combine two partitions in a partitioned table. The complete syntax for the `ALTER PARTITION FUNCTION` statement is as follows:

```
ALTER PARTITION FUNCTION partition_function_name(){SPLIT RANGE(boundary_value) |
MERGE RANGE (boundary_value)};
```

The `MERGE RANGE` clause combines two partitions, and the `SPLIT RANGE` clause splits a partition into two partitions. Each of these clauses specifies the boundary value for which the split or merge should occur. In this scenario, including the `MERGE RANGE (8000)` clause combines the partition to the left and right of the boundary value of 8000, and the table will be partitioned as follows:



Partition #	Values
1	<= 5000
2	5001 - 11000
3	11001 - 14000
4	14001 and above

You should not execute the statement that includes `SPLIT RANGE (8000)`. The `SPLIT RANGE` clause divides a partition into two separate partitions. It does not combine the partitions as required in this scenario. In addition, before you split partitions, you must ensure that the partition scheme has enough file groups to accommodate the new partition or an error is generated. You must also specify a value that is different from any other defined boundary value.

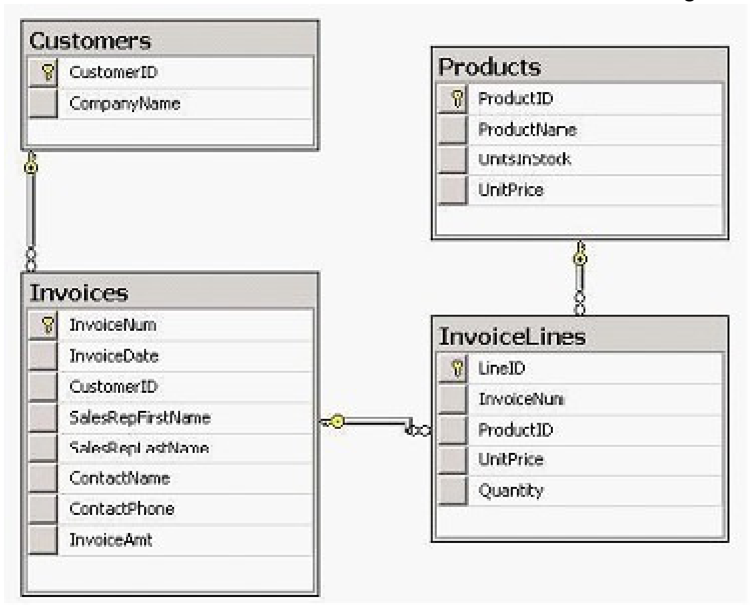
You should not execute the statement that includes `SPLIT RANGE (8000, 11000)`. This statement generates a syntax error because only a single boundary value may be specified with the `SPLIT RANGE` and `MERGE RANGE` clauses.

You should not execute the statement that includes `MERGE RANGE (8001)`. This statement will generate the following error because 8001 is not a valid boundary value:

```
Msg 7715, Level 16, State 1, Line 1The specified partition range value could not
be found.
```

QUESTION 29

Your company records information about its products, sales transactions, and customers in a SQL Server 2008 database. The current data model was created using Transact-SQL statements:



Consider the following requirements:
Each customer is assigned a company sales representative who is responsible for managing the customer's account.
Each customer is a company that has a designated contact person with whom the assigned sales representative communicates.
Each customer may place multiple orders for multiple products.
The unit price of each product may increase or decrease over time.
Each customer placing an order for a product must be charged the current unit price for the product.

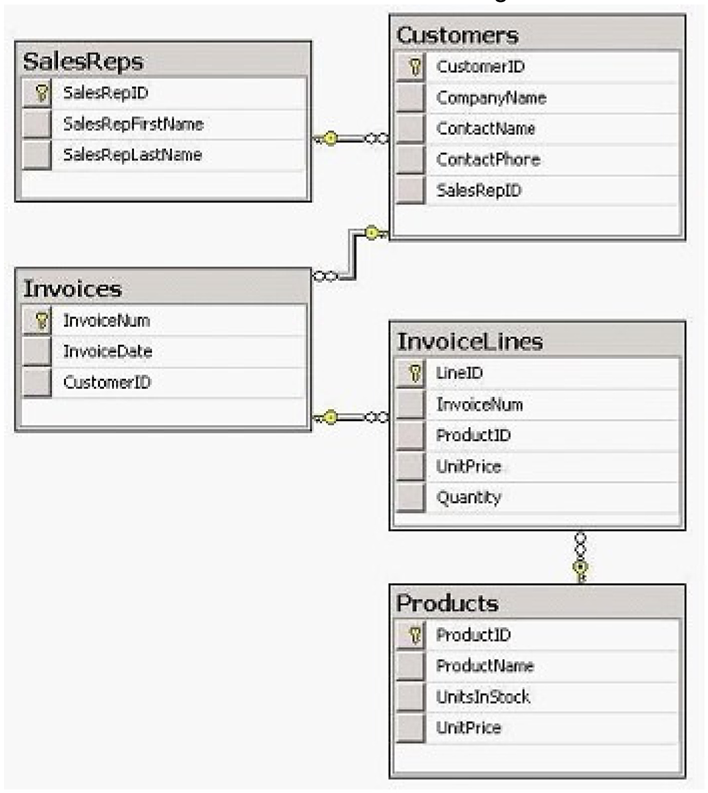
You created the **SalesReps** table using the following Transact-SQL:
```
CREATE TABLE SalesReps (SalesRepID int IDENTITY(1,1) PRIMARY KEY,
SalesRepFirstName varchar(25),SalesRepLastName varchar(30));
```
You must make additional modifications to the data model to meet the requirements and normalize the database to third normal form. The data model should support minimal redundant data.
Which additional Transact-SQL should you execute?

A.
```
ALTER TABLE Customers ADD ContactName varchar(50),ContactPhone char(14),
SalesRepID int REFERENCES SalesReps(SalesRepID);
ALTER TABLE Invoices DROP COLUMN SalesRepFirstName,SalesRepLastName,
ContactName,ContactPhone,InvoiceAmt;
```
B.
```
ALTER TABLE Customers ADD
ALTER TABLE Invoices ADD SalesRepID int REFERENCES SalesReps(SalesRepID);
ALTER TABLE Invoices DROP COLUMN SalesRepFirstName,SalesRepLastName,
ContactName,ContactPhone,InvoiceAmt;
```
C.
```
ALTER TABLE Customers ADD ContactName varchar(50),ContactPhone char(14),
SalesRepID int REFERENCES SalesReps(SalesRepID);
ALTER TABLE Invoices DROP COLUMN SalesRepFirstName,
SalesRepLastName,ContactName,ContactPhone,InvoiceAmt;
ContactName varchar(50),ContactPhone char(14),SalesRepID int REFERENCES
SalesReps(SalesRepID);
```
D.
```
ALTER TABLE Customers ADD
ALTER TABLE Invoices DROP COLUMN SalesRepFirstName,SalesRepLastName,
ContactName,ContactPhone,InvoiceAmt;
ALTER TABLE InvoiceLines DROP COLUMN UnitPrice;
```

Answer: A

Section: (none)

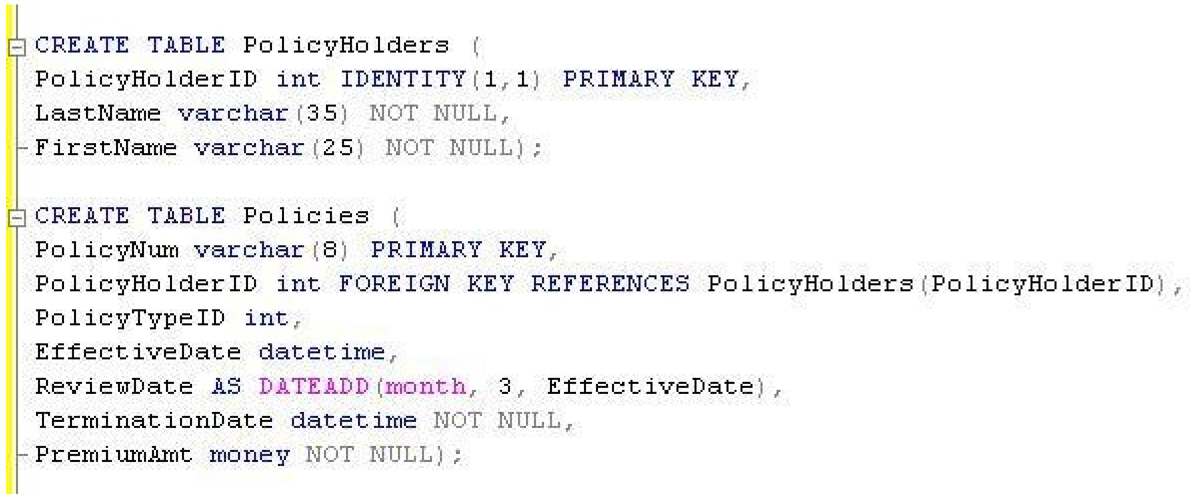
Explanation/Reference:
The data model will resemble the following:



To avoid storing redundant data, each table should be designed to store only data for a single entity. Foreign keys can then represent the relationships between the entities. There are formal rules, referred to as normal forms, which define how data in a data model should be separated. A database that complies with the normal forms is referred to as normalized to a normal form. Normalization to the third normal form is usually sufficient. A database complies with first normal form if no table has columns that define similar attributes and if no column contains multiple values in a single row. A database complies with second normal form if it complies with first normal form, and if each column that is not part of a primary key depends on all of the columns of the primary key in that table, and not on a subset of the primary key's columns. A database complies with third normal form if it complies with second normal form and if, in each table, columns that are not covered by the primary key do not depend on each other.
The data model in this scenario complies only with second normal form. In the **Invoices** table, the **ContactName** and **ContactPhone** columns depend on the **CustomerID** column, and the **SalesRepFirstName** column depends on the **SalesRepLastName** column; neither **CustomerID** nor **SalesRepLastName** is a part of the primary key in the **Invoices** table. To normalize the database to third normal form, you should remove the information about sales representatives and customer contacts from the **Invoices** table. You should create a new table, **SalesReps**, and move the **SalesRepFirstName** and **SalesRepLastName** columns to it. In the **Invoices** table, you should add a **SalesRepID** column that will reference the **SalesRepID** primary key column in the **SalesReps** table. With this modification, the name of each sales representative will be recorded only once in the **SalesReps** table rather than being recorded for each customer invoice.
The **ContactName** and **ContactPhone** columns should be removed from the **Invoices** table and added to the **Customers** table because each customer, not each invoice, is associated with a single contact.
The **InvoiceAmt** column is also redundant and can be removed without losing any important information. The **InvoiceAmt** column in the **Invoices** table represents the total amount of an invoice. These totals can be calculated from the unit prices and quantities of the products sold for each invoice. Therefore, you should remove the **InvoiceAmt** column from the **Invoices** table to minimize redundant data.
You should not use the Transact-SQL that adds the **SalesRepID** column to the **Invoices** table. This would

associate each invoice with a specific sales representative. In this scenario, you wanted to associate each customer with a specific sales representative.
You should not use the Transact-SQL that does not drop the **InvoiceAmt** column from the **Invoices** table. This column can be calculated in a query as needed by multiplying the **UnitPrice** and **Quantity** columns in the **InvoiceLines** table and summing the extended prices for the order. Therefore, this column should be omitted from the data model.
You should not use the Transact-SQL that drops the **UnitPrice** column from the **InvoiceLines** table. Although this column might initially seem redundant, it must be included in the **InvoiceLines** table to capture the current unit price at the time of each order. The **UnitPrice** column in the **Products** table contains the product's current unit price, and the **UnitPrice** column in the **InvoiceLines** table contains the unit price at the time the order was placed.

QUESTION 30
You are a database developer on an instance of SQL Server. You executed the following statements to create the **Policies** and **PolicyHolders**
tables:



```
CREATE TABLE PolicyHolders (
PolicyHolderID int IDENTITY(1,1) PRIMARY KEY,
LastName varchar(35) NOT NULL,
FirstName varchar(25) NOT NULL);

CREATE TABLE Policies (
PolicyNum varchar(8) PRIMARY KEY,
PolicyHolderID int FOREIGN KEY REFERENCES PolicyHolders(PolicyHolderID),
PolicyTypeID int,
EffectiveDate datetime,
ReviewDate AS DATEADD(month, 3, EffectiveDate),
TerminationDate datetime NOT NULL,
PremiumAmt money NOT NULL);
```

You need to ensure that the following conditions are met when a policy is added or updated in the **Policies** table:
- Each policy must have an effective date.
- Each policy must be associated with a single policy holder.
- Each policy must have a premium amount that is numeric and greater than zero.
- Each policy must have a review date that is before the termination date.

Which Transact-SQL should you execute?

A.
```
ALTER TABLE dbo.Policies
ADD CONSTRAINT nn_EffectiveDate NOT NULL,
CONSTRAINT ck_PremiumAmt CHECK (PremiumAmt > 0),
CONSTRAINT ck_ReviewDate CHECK (ReviewDate < TerminationDate);
GO
```

B.
```
ALTER TABLE dbo.Policies
DROP COLUMN ReviewDate;
GO
ALTER TABLE dbo.Policies
ALTER COLUMN EffectiveDate datetime NOT NULL;
GO
ALTER TABLE dbo.Policies
ADD ReviewDate AS DATEADD(month, 3, EffectiveDate) PERSISTED;
GO
ALTER TABLE dbo.Policies
ADD CONSTRAINT ck_PremiumAmt CHECK (PremiumAmt > 0),
CONSTRAINT ck_ReviewDate CHECK (ReviewDate < TerminationDate);
GO
```
C.
```
ALTER TABLE dbo.Policies
ALTER COLUMN EffectiveDate datetime NOT NULL;
GO
ALTER TABLE dbo.Policies
ALTER COLUMN ReviewDate AS DATEADD(month, 3, EffectiveDate) PERSISTED;
GO
ALTER TABLE dbo.Policies
ADD CONSTRAINT ck_PremiumAmt CHECK (PremiumAmt > 0),
CONSTRAINT ck_ReviewDate CHECK (ReviewDate < TerminationDate);
GO
```
D.
```
ALTER TABLE dbo.Policies
ALTER COLUMN EffectiveDate datetime NOT NULL;
GO
ALTER TABLE dbo.Policies
ADD CONSTRAINT ck_PremiumAmt CHECK (PremiumAmt > 0),
CONSTRAINT ck_ReviewDate CHECK (ReviewDate < TerminationDate);
GO
```

Answer: B
Section: (none)

Explanation/Reference:
The requirement that each policy be associated with a single policy holder was met when the **Policies** table was created, because the
PolicyHolderID column has a `FOREIGN KEY` constraint that references the **PolicyHolderID** in the **PolicyHolders** table. A `FOREIGN KEY`
constraint limits the values in the foreign key column to the values that exist in the referenced column. The `REFERENCES` clause specifies the
referenced column, which in this scenario is the **PolicyHolderID** column in the **PolicyHolders** table. This relates each policy in the **Policies** table to
the parent row in the **PolicyHolders** table.
To meet the other requirements, you should first drop the **ReviewDate** column from the **Policies** table and re-create it using the `PERSISTED`
keyword. In this scenario, you need to ensure that each policy's review date is before the termination date. To do so, you need to create a `CHECK`
constraint on the **ReviewDate** column. A `CHECK` constraint is used to restrict the data that is allowed for a column to specific values. A `CHECK`
constraint consists of a Boolean expression that evaluates to either `TRUE` or `FALSE`. If the expression evaluates to `TRUE`, the value is allowed for the
column, and if the expression evaluates to `FALSE`, the value is not allowed for the column. However, in this scenario, the **ReviewDate** column is a
computed column that was defined without the `PERSISTED` keyword. To create a `CHECK` constraint on the **ReviewDate** column, you must make the
computed column persisted by dropping the column and re-creating the column using the `PERSISTED` keyword.

You are also required to ensure that each policy has an effective date. To do so, you can create a `NOT NULL` constraint on the **EffectiveDate**
column. A `NOT NULL` constraint specifies that a value other than `NULL` must be specified for a column. You must perform this action after you drop
the **ReviewDate** column, but before you re-create it, because you cannot alter the **EffectiveDate** column to add a `NOT NULL` constraint if there is a
computed column that references the **EffectiveDate** column.
Finally, you should create the `CHECK` constraints to ensure that the premium amount is greater than zero and that the review date is before the
termination date. You should note that the **money** data type for the **PremiumAmt** column will ensure that the premium amount is numeric.
You should not use the Transact-SQL that issues two `ALTER TABLE` statements, one to create a `NOT NULL` constraint on the **EffectiveDate**
column and one to add constraints. The first `ALTER TABLE` statement will fail because you cannot add a `NOT NULL` constraint, because the
EffectiveDate column is referenced by the **ReviewDate** computed column expression. The second statement will fail because you cannot create a
constraint on a computed column that is not persisted. The following error messages will be displayed:

```
Msg 5074, Level 16, State 1, Line 4
The column 'ReviewDate' is dependent on column 'EffectiveDate'.
Msg 4922, Level 16, State 9, Line 4
ALTER TABLE ALTER COLUMN EffectiveDate failed because one or more objects access
this column.
Msg 1764, Level 16, State 1, Line 1
Computed Column 'ReviewDate' in table 'Policies' is invalid for use in 'CHECK
CONSTRAINT' because it is not
persisted.
Msg 1750, Level 16, State 0, Line 1
Could not create constraint. See previous errors.
```

You should not use the Transact-SQL that issues three `ALTER TABLE` statements, one to create a `NOT NULL` constraint on the **EffectiveDate**
column, one to alter the **ReviewDate** column, and one to add constraints. The first `ALTER TABLE` statement will fail because you cannot add the
`NOT NULL` constraint, because the **EffectiveDate** column is referenced by the **ReviewDate** computed column expression. The second statement
will fail because you cannot use the `ALTER COLUMN` clause of the `ALTER TABLE` to make an existing computed column persisted. To make an
existing computed column persisted, you must drop the computed column and re-create it.
The third statement fails because you cannot create a `CHECK` constraint on a computed column that is not persisted. The following error messages
will be displayed:

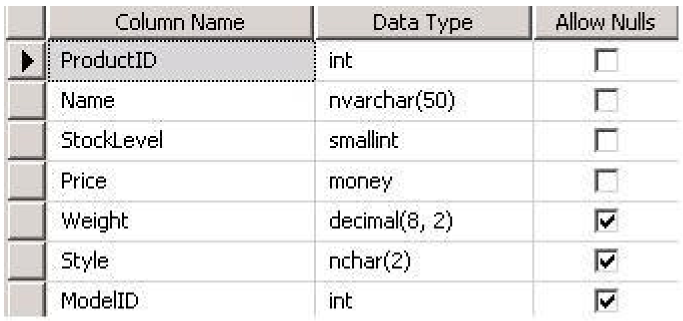
```
Msg 5074, Level 16, State 1, Line 4
The column 'ReviewDate' is dependent on column 'EffectiveDate'.
Msg 4922, Level 16, State 9, Line 4
ALTER TABLE ALTER COLUMN EffectiveDate failed because one or more objects access
this column.
Msg 156, Level 15, State 1, Line 2
Incorrect syntax near the keyword 'AS'.
Msg 1764, Level 16, State 1, Line 1
Computed Column 'ReviewDate' in table 'Policies' is invalid for use in 'CHECK
CONSTRAINT' because it is not
persisted.
Msg 1750, Level 16, State 0, Line 1
Could not create constraint. See previous errors.
```

You should not use the Transact-SQL that issues one `ALTER TABLE` statement because this code will generate the following error:

```
Msg 156, Level 15, State 1, Line 5
Incorrect syntax near the keyword 'NOT'.
```

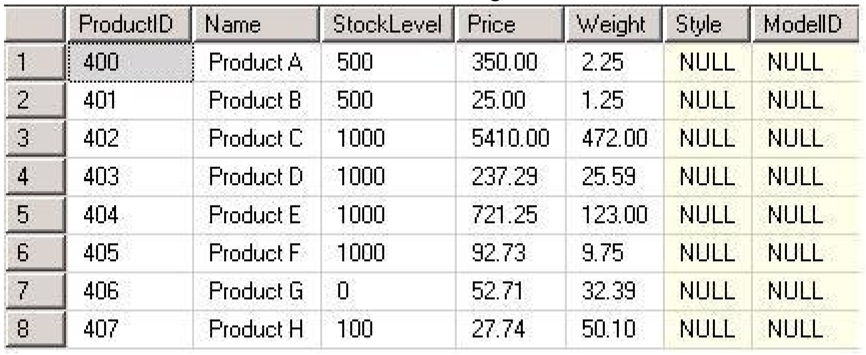
`NOT NULL` constraints are not added to a column using the `ADD CONSTRAINT` clause of the `ALTER TABLE` statement. You add a `NOT NULL`
constraint to an existing column using the `ALTER COLUMN` clause. Even if this constraint were removed from the statement, the statement

QUESTION 31
You are a database administrator on an instance of SQL Server 2008. Your **Products** table is defined as follows:



Column Name	Data Type	Allow Nulls
ProductID	int	☐
Name	nvarchar(50)	☐
StockLevel	smallint	☐
Price	money	☐
Weight	decimal(8, 2)	☑
Style	nchar(2)	☑
ModelID	int	☑

The **Products** table contains the following data:



	ProductID	Name	StockLevel	Price	Weight	Style	ModelID
1	400	Product A	500	350.00	2.25	NULL	NULL
2	401	Product B	500	25.00	1.25	NULL	NULL
3	402	Product C	1000	5410.00	472.00	NULL	NULL
4	403	Product D	1000	237.29	25.59	NULL	NULL
5	404	Product E	1000	721.25	123.00	NULL	NULL
6	405	Product F	1000	92.73	9.75	NULL	NULL
7	406	Product G	0	52.71	32.39	NULL	NULL
8	407	Product H	100	27.74	50.10	NULL	NULL

Which Transact-SQL statement will execute successfully and modify the size of an existing column in the **Products** table?

A.
```
ALTER TABLE Products
ALTER COLUMN Weight decimal(3,2) NULL;
```
B.
```
ALTER TABLE Products
ALTER COLUMN Weight decimal(7,2) NULL;
```
C.
```
ALTER TABLE Products
ALTER COLUMN Style nchar(5) NOT NULL;
```
D.
```
ALTER TABLE Products
ALTER COLUMN Price decimal(5,2) NOT NULL;
```

Answer: B
Section: (none)

Explanation/Reference:
You can use the `ALTER COLUMN` clause of the `ALTER TABLE` statement to modify the size of an existing column in a table. However, the column
size cannot be made smaller than the largest data value that currently exists in the table.
The `ALTER TABLE` statement that resizes the **Price** column is incorrect because it tries to resize the column to a size smaller than the largest value
for the **Price** column. The statement will generate the following error:
```
Msg 8115, Level 16, State 8, Line 7
Arithmetic overflow error converting money to data type numeric.
```
The `ALTER TABLE` statement that resizes the **Style** column is incorrect because it contains a `NOT NULL`

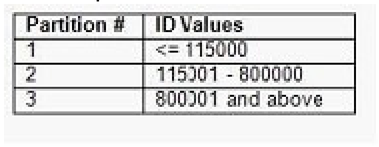
constraint. The **Products** table currently
contains `NULL` values for the **Style** column. Therefore, you cannot modify the column to include a `NOT NULL` constraint. The statement will
generate an error similar to the following:

```
Msg 515, Level 16, State 2, Line 4
Cannot insert the value NULL into column 'Style', table 'KIT3.dbo.Products';
column does not allow nulls.
UPDATE fails.
```

The `ALTER TABLE` statement that resizes the **Weight** column to a precision and scale of (`3,2`) is incorrect because it tries to resize the column to a
size that is smaller than the largest value for the **Weight** column. The statement will generate the following error:

```
ALTER TABLE dbo.MyProducts
ALTER COLUMN Weight decimal (3,2) NULL;
Msg 8115, Level 16, State 8, Line 5
Arithmetic overflow error converting numeric to data type numeric.
```

QUESTION 32
You are a database developer on an instance of SQL Server 2008. You have a partitioned table named **TransAudit** that contains historical audit records. Older audit records are rarely accessed. The **TransAudit** table is partitioned as follows:



Partition #	ID Values
1	<= 115000
2	115001 - 800000
3	800001 and above

You want to minimize storage required for the **TransAudit** table. Older audit records that reside in partition #1 were originally imported into the table from a legacy system using an SSIS package.
You analyze the underlying data and decide to implement row-level compression for partition #1. You also want to implement row-level and page-level compression on partition #2, but no compression on partition #3 because it is more frequently accessed.
You want to perform this task with minimal effort and resource usage.
Which Transact-SQL should you use?

A.
```
ALTER TABLE TransAudit
REBUILD PARTITION = 1 WITH (DATA_COMPRESSION=ROW);
ALTER TABLE TransAudit
REBUILD PARTITION = 2 WITH (DATA_COMPRESSION=BOTH);
```

B.
```
ALTER TABLE TransAudit
REBUILD PARTITION = 1 WITH (DATA_COMPRESSION=both);
ALTER TABLE TransAudit
REBUILD PARTITION = 2 WITH (DATA_COMPRESSION=PAGE);
```

C.
```
ALTER TABLE TransAudit REBUILD PARTITION ALL
WITH (DATA_COMPRESSION=ROW ON PARTITIONS(1),
DATA_COMPRESSION=PAGE ON PARTITIONS(2));
```

D.
```
ALTER TABLE TransAudit REBUILD PARTITION = 1 WITH (DATA_COMPRESSION=ROW);
ALTER TABLE TransAudit REBUILD PARTITION = 2 WITH (DATA_COMPRESSION=PAGE);
```

Answer: D
Section: (none)

Explanation/Reference:
You cannot use the Transact-SQL that specifies a `DATA_COMPRESSION` value of `BOTH` for partition #2. `BOTH` is not a valid value for the `DATA_COMPRESSION` setting. This statement will generate a syntax error. Valid values for the `DATA_COMPRESSION` setting are `NONE`, `PAGE`, and `ROW`.
You should not use the Transact-SQL that includes the `REBUILD PARTITION ALL` clause. Although it does

set the data compression settings as desired, it will rebuild all of the partitions. In this scenario, to minimize effort and resources, you are making no changes to partition #3. Therefore, there is no need to rebuild that partition.
You should not use the Transact-SQL that uses a single `ALTER TABLE` statement because in this scenario, you wanted to use different compression strategies for different partitions of the table.

QUESTION 33
You are a database developer on an instance of SQL Server 2008. Your **Meeting** table was defined using the following Transact-SQL:

```
CREATE TABLE Meeting (ID int IDENTITY(1,1) PRIMARY KEY,Description varchar(30)
NOT NULL,MaxAttendance smallint DEFAULT 0,Type bit NULL,Priority tinyint CHECK
(Priority BETWEEN 0 and 10),Cost money NULL);
```

You create a view on the **Meeting** table using the following statement:

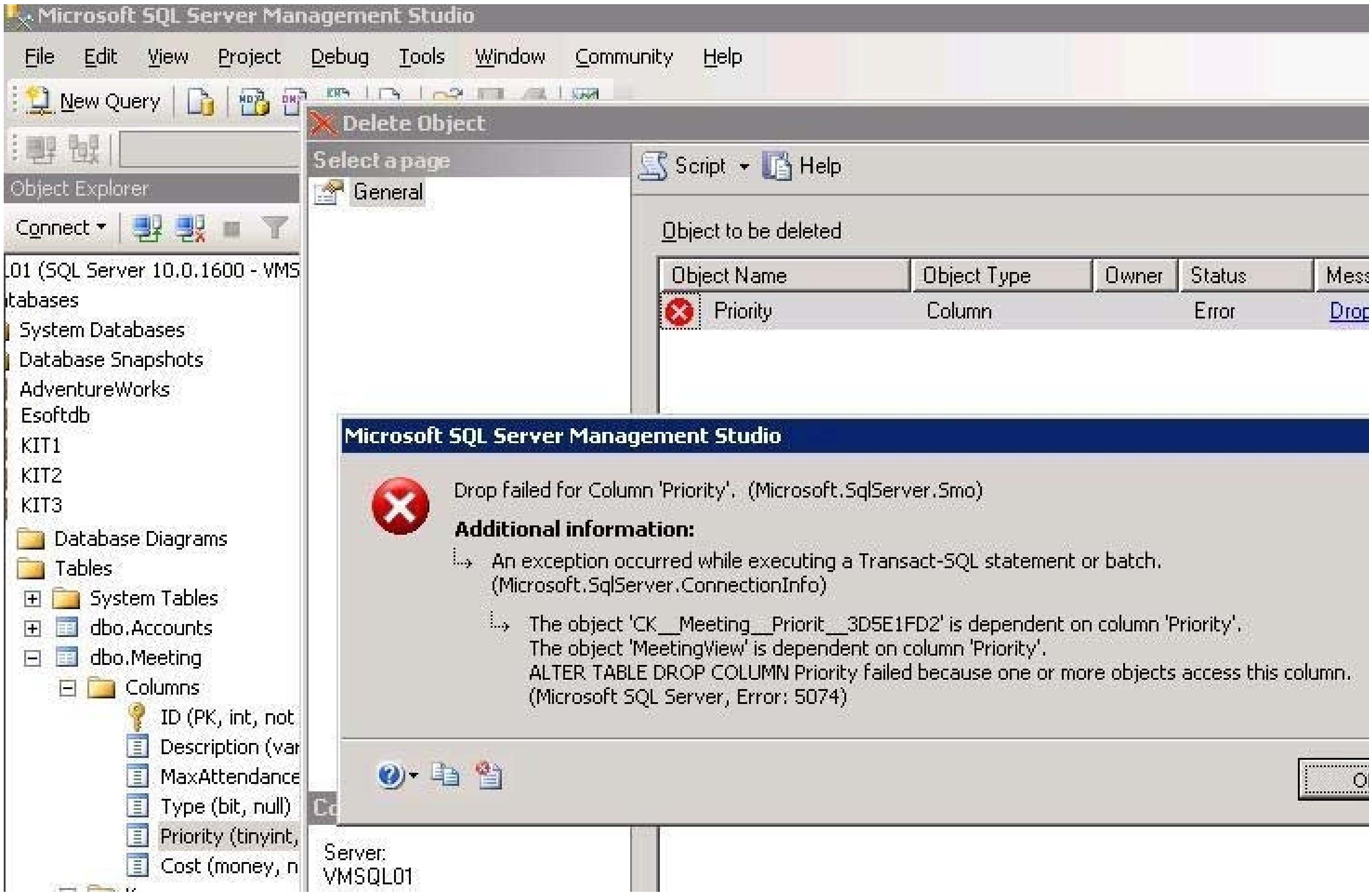
```
CREATE VIEW MeetingViewWITH SCHEMABINDING AS SELECT ID, Description, Priority
FROM dbo.MeetingWHERE Priority BETWEEN 1 and 5;
```

You no longer need the **Priority** column and want to drop it from the **Meeting** table.
Which action should you take?

A. Alter the view to remove the `WHERE` clause from the view's definition.
B. Remove the `CHECK` constraint on the **Priority** column and alter the view to remove all references to the **Priority** column.
C. Drop and re-create the view without the `SCHEMABINDING` option.
D. Move the view to another schema.

Answer: B
Section: (none)

Explanation/Reference:
To drop the **Priority** column from the **Meeting** table, you should remove the `CHECK` constraint on the **Priority** column and alter the view to remove all references to the **Priority** column. When you created the view, you included the `WITH SCHEMABINDING` clause. The `WITH SCHEMABINDING` clause ensures that base tables of a view cannot be dropped or modified in a way that affects the view's definition. This prevents users from dropping or modifying base tables in a way that makes the view unusable. To drop base tables or make such modifications, you would need to first drop the view, alter the view omitting `SCHEMABINDING`, or alter the view to remove any unwanted dependencies. In this scenario, the **Priority** column also has a `CHECK` constraint, which must also be removed before dropping the column or the following error occurs:



Microsoft SQL Server Management Studio

Drop failed for Column 'Priority'. (Microsoft.SqlServer.Smo)

Additional information:

An exception occurred while executing a Transact-SQL statement or batch. (Microsoft.SqlServer.ConnectionInfo)

The object 'CK__Meeting__Priorit__3D5E1FD2' is dependent on column 'Priority'.
The object 'MeetingView' is dependent on column 'Priority'.
ALTER TABLE DROP COLUMN Priority failed because one or more objects access this column. (Microsoft SQL Server, Error: 5074)

All of the other options are incorrect because these actions will not allow you to drop the **Priority** column from the **Meeting** table. To drop the column successfully, you must remove the column's `CHECK` constraint and then perform one of two actions: drop and re-create the view without the `SCHEMABINDING` option, or remove all of the column's references from the view.

QUESTION 34
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You use SQL Server 2008 to create a solution. You intend to import data from an external source into a table.
You must make sure that the tasks below are achieved:
- The rows that fail the foreign key constraints during import are inserted into a separate table.
- Even if the import encounters rows that fail foreign key constraints, it is successfully completed.

So what should you do?

A. An AFTER trigger has to be used
B. CHECK constraints have to be used
C. An INSTEAD OF trigger has to be used
D. During the import process disable the foreign keys

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 35
You manage a database, **Prod**, on an instance of SQL Server 2008. You have a partitioned table, **TransDetails**, which was defined using the following Transact-SQL:
```
CREATE PARTITION FUNCTION PFunction1 (int)AS RANGE LEFT FOR VALUES (2000, 3000);
CREATE PARTITION SCHEME PScheme1 AS PARTITION PFunction1 TO (fg1, fg2, fg3);
CREATE TABLE TransDetails (TransID int,Description varchar(50),Region varchar
(20),Date datetime,TypeID int,EmpID int,Status bit)ON PScheme1(TypeID);
```
You open a session and execute the following Transact-SQL:
```
ALTER TABLE TransDetails SET (LOCK_ESCALATION = TABLE);GO BEGIN TRANSACTION
UPDATE TransDetails SET TypeID = TypeID + 1000 WHERE TypeID > 4000;GO
```
Which type of lock will be acquired if lock escalation occurs?

A. an Exclusive (X) lock on the **TransDetails** table
B. an Update (U) lock on the **TransDetails** table
C. a partition-level lock on one of the **TransDetails** table's partitions
D. an Intent Exclusive (IX) lock on the **TransDetails** table

Answer: A
Section: (none)

Explanation/Reference:
Currently, an Exclusive (X) lock will be acquired on the **TransDetails** table if lock escalation occurs. In this scenario, you issued an `ALTER TABLE` statement that set `LOCK_ESCALATION` for the **TransDetails** table to `TABLE`, which is actually the default. Normally, table-level lock escalation is fine. However, with partitioned tables, you may not want to lock the entire table, but only a single partition, such as the partition containing the rows being updated in this scenario. This would allow queries against other partitions in the table to execute successfully without being blocked. By default, SQL Server will escalate locks to the table level as needed. However, to override this default behavior, or to disable lock escalation altogether, you can set the `LOCK_ESCALATION` setting for a table. The `LOCK_ESCALATION` setting can have one of the following values:
`TABLE`: Lock escalation is performed to the table level, which is the default.
`AUTO`: Lock escalation is performed to the table level for non-partitioned tables, and to the partition level for partitioned tables.
`DISABLE`: Lock escalation is disabled, and lock escalation does not typically occur. SQL Server only escalates locks in specific situations where it is absolutely required to ensure data integrity.

In this scenario, you could instruct SQL Server to escalate locks to the partition level only using the following statement:
```
ALTER TABLE TransDetails SET (LOCK_ESCALATION=AUTO);
```
Doing so would prevent lock escalation to the table level, and allow other queries to execute successfully on other unaffected partitions of the table. To identify the `LOCK_ESCALATION` setting for a table, you can query **sys.tables**. For example, the following statement would display the current `LOCK_ESCALATION` setting for the **TransDetails** table:
```
SELECT lock_escalation_desc FROM sys.tables WHERE name = 'TransDetails';
```
All of the other options are incorrect because an exclusive lock will be held on the **TransDetails** table because it is a partitioned table and the `LOCK_ESCALATION` setting is set to escalate locks to the table level.

QUESTION 36
You have a **Research** database. The database resides on a SQL Server 2008 database instance named **SRV1**.
You want users to be able to specify up to 10 string values. Using these specified values, you want to perform complex string comparisons and rank each of the strings with a rating value. Researchers need to store these

string values and their corresponding ratings in the **Scenario** table in the **Research** database. You want to provide this functionality while optimizing performance and minimal development. Which mechanism should you use to implement this functionality?

A. an extended stored procedure
B. a CLR table-valued function
C. a Transact-SQL multi-statement table-valued function that uses the `OVER` clause
D. a CLR user-defined aggregate

Answer: B
Section: (none)

Explanation/Reference:
A CLR function is created in a .NET Framework language and then registered in the database so it can be accessed using Transact-SQL. You should consider using CLR functions or procedures when you need to perform CPU-intensive processing, or if the desired functionality can be implemented more efficiently using . NET Framework classes. The .NET Framework offers more robust string comparison and manipulation functionality that is not natively supported in SQL Server. CLR functions can return a single scalar value or a result set. Because the function would need to return more than one value, it would be created as a CLR table-valued function that returns a result set to the caller. The data could then be easily inserted into the **Scenario** table as required.
You should not use an extended stored procedure because this would require additional development effort.
You should not use a Transact-SQL multi-statement table-valued function that uses the `OVER` clause. Although you could use a Transact-SQL function to perform complex string comparisons, a CLR function would be a better choice because it offers more robust built-in methods that will optimize performance and reduce the coding complexity. The `OVER` clause specifies the order of the rowset or the partitioning order before an associated windowing function is applied.
You should not use a CLR user-defined aggregate. A user-defined aggregate would be used to implement a customized aggregate function. In this scenario, there were no aggregation requirements.

QUESTION 37
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. Large image files will be stored in a table. The files are about 20-50 MB in size. The requirements below have to be met:
The image files are part of the database backup; The image files are accessible by applications that use Win32 APIs. You have to store the images files by using a suitable strategy. So which strategy should you use?

A. An image data type should be used
B. The varbinary(max) data type should be used
C. The varbinary(max) data type should be used along with the FILESTREAM attribute
D. The image file should be stored in a file system. Then you store the file location in the database by using a varchar data type

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 38
You have a table that has 10 million rows. The table has a view that returns all of the rows. You discover performance degradation when you run an unfiltered view. You need to recommend a solution to replace the

view. The solution must require that the returned data is filtered by a parameter.
What should you use?

A. an indexed view
B. a scalar function
C. a table-valued function
D. a table-valued type

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 39
You are designing a document repository application that will contain over 100,000 documents. The repository will have the following specifications:
·
Documents can be associated to 30 different properties ·
Most documents will have less than 10 properties defined You need to design a single table for the application. The solution must use the minimum the amount of storage space.

What should you include in the design?

A. an XML data type
B. nvarchar() null
C. sparse columns
D. varchar(max) not null

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 40
You are a database developer on an instance of SQL Server 2008. Your **Sales** database contains sales and marketing information for your company. You receive an XML file in the following format that contains leads from sales representatives in the field:
```
<LeadData>
  <Salesperson ID="45">
    <DailyLeads>
      <Leads ID="22" InitDate="2009-01-07" >
       <Company Name="NuTex Corporation" EstRev="40000" />
       <Company Name="Verigon Inc." EstRev="125000" />
      </Leads>
    </DailyLeads>
  </Salesperson>
  <Salesperson ID="82">
    <DailyLeads>
      <Leads ID="29" InitDate="2009-02-12" >
       <Company Name="VisionWorx" EstRev="500000" />
       <Company Name="InterConn" EstRev="187000" />
      </Leads>
    </DailyLeads>
  </Salesperson>
</LeadData>
```

You load the XML into an **xml** variable named `@x` and execute the following Transact-SQL:

```
EXEC sp_xml_preparedocument @idoc OUTPUT, @x
SELECT * INTO dbo.SalesLeads
 FROM OPENXML(@idoc, 'LeadData/Salesperson', 1)
 WITH (ID int, Leads xml 'DailyLeads');
EXEC sp_xml_removedocument @idoc;
```

What will be the result?

A. An XML document is created containing each salesperson's leads.
B. A single row is inserted into the **SalesLeads** table.
C. Multiple rows are inserted into the **SalesLeads** table.
D. The statement fails because the `OPENXML` function is incorrectly specified.

Answer: C
Section: (none)

Explanation/Reference:
In this scenario, you first call the **sp_xml_preparedocument** stored procedure to create an internal handle to the XML document. Then, you include a query that contains the `OPENXML` function in the `FROM` clause. The `OPENXML` function is used to extract data from an XML document and present it as a relational rowset. This function can be used in `SELECT` statements where a table or view would be specified to extract data from an XML document into relational format. The `OPENXML` function accepts the following parameters:
`idoc`: Identifies the integer document handle returned using the **sp_xml_preparedocument** system stored procedure.
`rowpattern`: Specifies the name of the node from which processing should begin.
`flag`: Identifies the type of mapping as either attribute-specific or element-specific. A `flag` value of 1 indicates that the mapping should be attribute-specific, and a value of 2 indicates that the mapping should be element-specific.
`colpattern`: Specifies an XPath pattern to identify the XML elements that should be mapped. This parameter is used with the `WITH` clause of the query.

In this scenario, the row pattern `'LeadData/Salesperson'` specifies that processing should begin at the `<Salesperson>` node. The `flag`value of 1 indicates that mapping is attribute-specific. The `WITH` clause specifies the columns that should appear in the output. The columns will be output in the order they are specified. For each column, you should specify the column name and data type, and an optional XPath expression. If you already have a table that has the desired columns, you can specify a table name instead of individually naming the columns. In this scenario, you would specify the `ID` and `Leads` columns in the `WITH` clause. The **ID** column will contain the `ID` attribute value for each salesperson as an **int** data type. The **Leads** column is an **xml** data type. This will generate a rowset that contains one row per salesperson. The**Leads** column of each row will contain the XML for each `<Leads>` element for the salesperson. You also include an `INTO` clause in the query to insert the result set into a new **SalesLeads** table. In this scenario, the **Leads** column for the salesperson with an **ID** value of 82 would contain the following XML:

```
<DailyLeads><Leads ID="29" InitDate="2009-02-12"> <Company Name="VisionWorx"
EstRev="500000" /><Company Name="InterConn" EstRev="187000" /></Leads></
DailyLeads>
```

After the data has been extracted from the XML document, you should call the **sp_xml_removedocument** stored procedure to remove the internal handle and free resources.
The option that states an XML document is created containing each salesperson's leads is incorrect because this query returns data in relational format, rather than XML.
The option that states a single row is inserted into the **SalesLeads** table is incorrect. Because you specified that the extraction should begin with the `<Salesperson>` node, a row will be inserted for each salesperson.

The option that states the statement fails because the `OPENXML` function is incorrectly specified is incorrect. The statement is syntactically correct and will not generate an error.

QUESTION 41
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. A database contains a table named Employee_Vacation. The HR gives you an updated list of employee vacations used which is in an XML formatted file. The extract of the XML format is

written in the manner below.
<Company Name ="ABC Company Pvt Ltd">
<EmployeeLeave>
<Employee ID = "1" Name="Jim Reeves" />
<Leaves>
<Leave Date="2008-02-12" />
<Leave Date="2008-02-13" />
<Leave Date="2008-02-14" />
</Leaves>
</EmployeeLeave>
You plan to import the data and update the Employee_Vacation table. In order to check the vacations days that used by each people, you design a query to count the number. What should you do to make sure that you can accurately calculate the vacation days for each people?

A. An XQuery expression should be used along with the LET clause and the count function.
Return the count in XML format
B. ou should aggregate the number of vacation days for each employee, and then return the total count in XML format. And an XML index should be used
C. You should you return the count in XML format after you execute the Transact-SQL count function on the vacation days. Before you perform this, you should convert XML data into a standard table format by using the OPENXML function
D. You should you return the count in XML format after you aggregate the number of vacation days from the tables. Before you perform this, you should use an XQuery expression to write the information from XML format to a SQL Server table

Answer: A
Section: (none)

Explanation/Reference:

QUESTION 42
You are a database solutions architect. Your company plans to develop a solution by using a SQL Server 2008 instance.

The solution has the following business requirements:
- Import data from various data sources such as Microsoft Office Excel, Microsoft SQL Server 2000, Microsoft SQL Server 2005, and CSV files.
- Profile the source data before it is imported.
- Provide collaboration and offline capabilities to mobile users.
- Allow mobile users to use heterogeneous data stores.

You need to configure the appropriate SQL Server components to accomplish the business requirements. You want to achieve this goal by using the minimum amount of administrative effort.
Which two SQL Server components should you use? (Each correct answer presents part of the solution. Choose two.)

A. Analysis Services
B. Reporting Services
C. Integration Services
D. Notification Services
E. A CONTAINSTABLE function

Answer: CE
Section: (none)

Explanation/Reference:

QUESTION 43
You are a database administrator on an instance of SQL Server. You have users in many geographical regions.
You need to provide data access to all of your regional offices. Each regional office updates its own regional data. You want each regional office to be responsible for backing up and maintaining its own data. Each regional office has access to only its data. However, at the company headquarters, you need to be able to access data from all regions. Which action should you take?

A. At each regional office, add a linked server for a single server at company headquarters.
B. Use snapshot replication.
C. Use peer-to-peer replication.
D. Implement a distributed partitioned view.

Answer: D
Section: (none)

Explanation/Reference:
The server at each regional office would contain data for the individual region. Each regional office could administer its own server independently. This would offload processing and administration effort to the regional offices. You could still access the data collectively from the company headquarters using a distributed partitioned view. Partitioned views are used when you have similar data stored in multiple tables and want to create a view to allow access to all of the data as if it were stored in a single table. The tables referenced in the view can reside on the same server or on different servers, such as at different regional offices in this scenario. To create a distributed partitioned view at the company headquarters, you could add a linked server for each region's server, and then create a view to access the data from each region. The view definition would use the fully qualified name for each `SELECT` statement and the `UNION ALL` operator to combine the results.
You should not add a linked server at each regional office. This would allow each regional office to access data across the enterprise, but would not offload the processing and administration to each regional office, or allow the company headquarters to access data across all regional offices.
You should not use snapshot or peer-to-peer replication. Replication is used to provide access to data across the enterprise while allowing the data to be synchronized. In this scenario, regional users only need to access data for their region, so no synchronization would be required. If you needed data synchronization between all locations, you could implement transactional replication using the central subscriber model.

QUESTION 44
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. The database will contain a table. And the table includes a parent-child relationship to itself. Each child might also be a parent, and this relationship might exist up to 10 levels deep. Now you have to use a single Transact-SQL query to retrieve all levels. So what should you do?

A. Before you add a correlated subquery to get the remaining levels, a query should be written to return the first level
B. You should return the first level by the common0-table expression you create, after this, union back to the expression itself to get the remaining levels.
C. Before you use the CROSS JOIN operator to join the table back to itself to get the remaining levels, you should written a query to return the first level Certkey.com - Make You Succeed To Pass IT Exams Certkey 70-451
D. You should return the first level by the view you create, after this, join the table back to the view to get the remaining levels by using the FULL OUTER JOIN operator.

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 45
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. Order data is stored by managers in a database. A list of customers is required for each manager. The list must be sorted in the descending order of the order amount. A query is created to generate the list at the end of each month. What should you do to make sure that the query executes as quickly as possible?

A. Before you sort the order data by order amount, you should create a cursor that returns each manager
B. You should use a SELECT statement which uses the OVER clause to rank the customers by order amount for each manager.
C. You should sort the results by manager and then by order amount. Before you perform this, you should create a correlated subquery to return the order amount for each manager
D. You should sort the results by account manager and then by order amount. Before you perform this, you should use the CROSS APPLY clause to create a query to list each account manager after you create a table-valued function which returns the order amount for a specific account manager.

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 46
You are designing a database table for a content management system. Users will store images and videos in the database.You need to ensure that the database can store files that are 20 MB or less. The solution must minimize the amount of space required to store the data.
Which data type should you use?

A. binary(20)

B. varbinary(20)
C. varbinary(max)
D. XML

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 47
You are a database developer on an instance of SQL Server 2008. Your **Prod** database contains many parameterized queries, and you have set the `PARAMETERIZATION` option to `FORCED` for the **Prod** database. You have several stored procedures that execute a specific query. You want to use simple parameterization for the query, instead of forced parameterization.
Which action should you take?

A. Call the **sp_get_query_template** stored procedure and use the result to create a template plan guide.
B. Set the `PARAMETERIZATION` database setting to `SIMPLE`.
C. Create a SQL plan guide for the query.
D. Modify the `PARAMETERIZATION` option for the query.

Answer: A
Section: (none)

Explanation/Reference:
. Template plan guides are used to override the `PARAMETERIZATION` database setting for specific groups of queries. If the database is using forced parameterization, you can create a template plan guide to use simple parameterization, or vice versa. First, you should call the **sp_get_query_template** stored procedure to obtain the parameterized version of the query. This will return the query string and the parameters used in parameterization. Then, you should create a template plan guide that includes the `PARAMETERIZATION FORCED` query hint. To create the plan guide, you use the **sp_create_plan_guide** system stored procedure. For example, if you previously used the **sp_get_query_template** system stored procedure for the specified query to return the `@stmt` and `@params` variables, you could create a template plan guide using the following statement:
```
EXEC sp_create_plan_guideN'MyTemplateGuide',@stmt,N'TEMPLATE',NULL,@params,
N'OPTION(PARAMETERIZATION FORCED)';
```
You should not create a SQL plan guide for the query. SQL plan guides are used with stand-alone Transact-SQL statements and batches.
You should not set the `PARAMETERIZATION` database setting to `SIMPLE`. This would cause all queries in the **Prod** database to use simple parameterization.
You should not modify the `PARAMETERIZATION` option for the query. The `PARAMETERIZATION` setting is specified at the database level, not the query level.

QUESTION 48
You have a stored procedure that uses a cursor. The stored procedure updates several related tables. You discover that the stored procedure runs slowly and uses a significant amount of resources on the database server.
You need to recommend changes to the stored procedure to meet the following requirements:
Minimize execution time
Minimize development effort
Minimize server resource requirements
What should you recommend?

A. Change the cursor to a dynamic cursor.
B. Change the cursor to a client-side cursor.
C. Rewrite the cursor by using set-based operations.
D. Rewrite the cursor by using recursive CLR stored procedure.

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 49
You are a database developer.
You create a database by using SQL Server 2008 in an enterprise environment.
The database contains two stored procedures named ModifySales and RetrieveSalesTrend.
The ModifySales stored procedure uses a transaction that updates a table named SalesOrders.
The RetrieveSalesTrend stored procedure retrieves and aggregates data from the SalesOrders table for a sales trend analysis report.
Both stored procedures are executed frequently each day.
Users report a considerable wait time while they run the sales trend analysis report.
You need to ensure that sales trend analysis report runs as quickly as possible.
What should you do?

A. Change the isolation level to SERIALIZABLE for ModifySales.
B. Change the isolation level to READ UNCOMITTED for ModifySales.
C. Add the NOWAIT hint to the SELECT statement in RetrieveSalesTrend.
D. Add note, the SELECT statements in RetrieveSalesTrend the NOLOCK.

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 50
You have a table that contains 5 million rows. The table has a clustered index. You plan to add an additional index on a column that contains 80 percent null values. The column is used extensively in WHERE clauses.
You need to implement an index strategy for the column. The solution must minimize the amount of storage space required for the index.
Which type of index should you use?

A. clustered
B. filtered
C. nonclustered
D. unique

Answer: B
Section: (none)

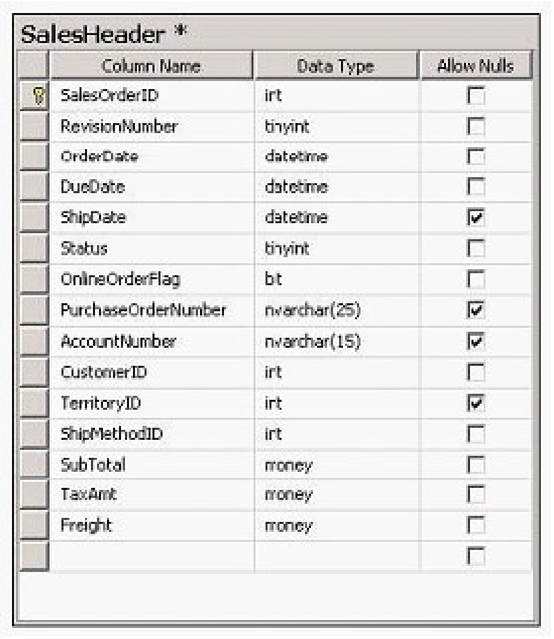
Explanation/Reference:

QUESTION 51
You manage SQL Server 2008 instances for a large retail organization. You want to design a reporting solution to allow users read-only access to data in databases on different instances.
Users generating reports only need read-only access. Users in different locations have different uses for the data. Some users perform complex reporting that is resource-intensive.
You want to provide reporting capability with optimum responsiveness for all users, while minimizing administration effort and performance impact to production systems using the data.
Which action should you take?

A. Configure peer-to-peer replication.
B. Create a database snapshot
C. Implement scalable shared databases.
D. Configure log shipping.

Answer: C
Section: (none)

Explanation/Reference:
Scalable shared databases allow you to distribute multiple read-only databases to multiple instances for read-only reporting.
You should not configure peer-to-peer replication. In this scenario, there was no requirement that reporting users have the most up-to-date information. You could use peer-to-peer replication to scale out a heavily accessed database. This would allow users to access consistent data and provide multiple database copies to be updated by different users.
You should not create a database snapshot. A database snapshot is a copy of a database as of single point in time.
You should not configure log shipping. With log shipping, transaction log backups from a primary database are automatically transferred to another database and then applied.
You are a database developer on an instance of SQL Server 2008. Your **Sales** database contains both current and historical sales data for your company. Your **SalesHeader** table is defined as follows:


SalesHeader *

Column Name	Data Type	Allow Nulls
SalesOrderID	int	☐
RevisionNumber	tinyint	☐
OrderDate	datetime	☐
DueDate	datetime	☐
ShipDate	datetime	☑
Status	tinyint	☐
OnlineOrderFlag	bit	☐
PurchaseOrderNumber	nvarchar(25)	☑
AccountNumber	nvarchar(15)	☑
CustomerID	int	☐
TerritoryID	int	☑
ShipMethodID	int	☐
SubTotal	money	☐
TaxAmt	money	☐
Freight	money	☐
		☐

QUESTION 52
You maintain a database named **Sales** on a SQL Server 2008 instance named **SQL1**. This SQL server is located at your company's main office in
Chicago. The **Sales** database contains sales and inventory-related data for your company.
Your company recently acquired another company that maintains its sales and inventory data in an Oracle database. You want employees of the
acquired company to continue to use Oracle database tools to maintain and access sales and inventory-related data.
On a weekly basis, you need to produce historical sales reports that query and aggregate data from both databases. The following requirements
must also be met:
 Authorized users at the main office must be able to run frequent queries against both databases to determine available inventory quantities.
 No employees of the recently acquired company should be able to access data stored at the main office.
Which action should you take?

A. Configure **SQL1** as a linked server using SQL Server Management Studio.
B. Configure the Oracle server as a linked server using the **sp_addlinkedserver** system stored procedure, and use distributed queries to
generate the weekly sales report and provide query access to users at the main office.
C. Create an SSIS package to extract data from the Oracle server and schedule the package to run weekly.
D. Create a query that uses the `OPENDATASOURCE` function and use SQL Server Agent to execute the query weekly to produce the weekly sales
report.

Answer: B
Section: (none)

Explanation/Reference:
Linked servers allow you to define a connection to a different
server and reuse the connection in Transact-SQL statements across multiple servers. The servers might be two SQL servers or other
heterogeneous data sources, as in this scenario. You can add a linked server using SQL Server Management Studio or using the
sp_addlinkedserver system stored procedure.
When you configure a linked server, you can have SQL Server connect to the remote server using the current user's SQL Server account or the
Windows credentials on the local server. You can also map SQL Server logins on the local server to logins on the remote server. After creating a
linked server definition, you can use the `OPENQUERY` function to execute a pass-thru query against the linked server's data sources, or you can
implement distributed queries that access data from multiple servers. In this scenario, distributed queries would allow users at the main office to
access data on **SQL1** and the Oracle server, and allow the weekly sales report to be generated.
You should not create a query that uses the `OPENDATASOURCE` function and use SQL Server Agent to execute the query weekly to produce the
weekly sales report. The `OPENDATASOURCE` function can be used to create an ad hoc connection and access remote data sources without defining
a linked server when specific data options are set to allow it. However, when using the `OPENDATASOURCE` function, the connection information and
credentials must be specified with each function call, and the `OPENDATASOURCE` function should only be used for infrequent ad hoc access. You
can also use the `OPENROWSET` function to access remote data, but `OPENDATASOURCE` and `OPENROWSET` should not be used for frequent access.
Because users at the main office must frequently execute queries against both data sources, you should configure the Oracle server as a linked

server to allow the users to execute the required distributed queries.
You should not create an SSIS package to extract data from the Oracle server and schedule the package to run weekly. SQL Server Integration
Services (SSIS) allows you to create packages to retrieve data from multiple data sources, transform and cleanse data, and load data into a variety
of data sources. You could use an SSIS package to extract data from the Oracle server and load it into a SQL Server database. However, in this
scenario, you did not want to transfer data from the Oracle server to **SQL1**.
You should not configure **SQL1** as a linked server using SQL Server Management Studio. When you configure a server as a linked server, data on
the server can be accessed remotely by other SQL Server instances. In this scenario, you did not need to provide remote users with access to data
stored on SQL1

QUESTION 53
You are a database developer on an instance of SQL Server 2008. You are creating a stored procedure that implements transactions. Your stored
procedure will be used by a production application to request supplies from inventory.
The application will allow users on the plant floor to query the parts available in the database from the **Product** table. Then, the application will allow
the user to select the product and the quantity of that product that should be delivered to the user's station. When the inventory request is finalized
by clicking the **Save** button in the application, the request is added as an internal pick order to be processed in the warehouse.
After a user clicks the **Save** button and the user is in the process of finalizing a request, no other users, internal or external, should be able to make
requests on the same product. You want to accomplish this with minimal locking.
Which action should you take?

A. Configure the application to use a transaction isolation level of `REPEATABLE READ`.
B. Specify the `TABLOCKX` table hint when initially querying the **Product** table.
C. Issue the `SET XACT_ABORT ON` statement at the beginning of the application.
D. Set the `ALLOW_SNAPSHOT_ISOLATION` database option to `OFF`.

Answer: A
Section: (none)

Explanation/Reference:
You should configure the application to use the `REPEATABLE READ` transaction isolation level. Isolation levels determine the degree of isolation for
a transaction from changes made to resources or data by other transactions. To protect data consistency, SQL Server uses locking. Locks protect
resources required by a transaction from being accessed or modified by other concurrent transactions. Unless you specify locking hints, SQL Server
automatically applies locks of the appropriate granularity, depending on the action being performed. The types of locks acquired depend on the
transaction isolation level. The `REPEATABLE READ` isolation level ensures that no other transaction can delete or update rows in the result set of
each `SELECT` statement in the current connection until the current transaction has either been committed or rolled back. Transactions using other
connections can insert new rows, which may appear in the result set if the same `SELECT` statement is reissued later in the current transaction.
In this scenario, insertion of new rows will not be a problem if users need to insert rows into the **Product** table while an existing product is being
requested. The following transaction isolation levels can be specified:

 SNAPSHOT: Ensures that the data read by a statement in a transaction will remain consistent until the transaction is complete. This isolation
level does not allow dirty reads, nonrepeatable reads, or phantom reads.
 SERIALIZABLE: Completely isolates transactions from one another. This isolation level is the most restrictive and generates the most locks.
 READ COMMITTED: Prevents statements from reading data that has been modified by other transactions but not committed. This isolation
level does not allow dirty reads, but does allow phantom reads and nonrepeatable reads, and is the default.
 REPEATABLE READ: Prevents statements from reading data that has been modified by other transactions but not yet committed, and also
prevents other transactions from modifying data read by the current transaction until the current transaction completes. This isolation level
does not allow dirty reads or nonrepeatable reads, but does allow phantom reads.
 READ UNCOMMITTED: Allows other transactions to read rows that have been modified by another transaction but not yet committed.
You should not specify the TABLOCKX table hint when initially querying the **Product** table. The TABLOCKX table hint is used to specify that a table
should be exclusively locked for a given statement. In this scenario, you want to minimize locking, and other users should be able to access
information for other products for which a request is not being made.
You should not issue the SET XACT_ABORT ON statement at the beginning of the application. The XACT_ABORT option determines how SQL
Server handles statements within transactions when runtime errors occur. When the XACT_ABORT option is set to ON and a Transact-SQL
statement raises an error at run time, the entire transaction is terminated and rolled back. When the XACT_ABORT option is set to OFF, only the
Transact-SQL statement that raised the error is rolled back.
You should not set the ALLOW_SNAPSHOT_ISOLATION database option to OFF. The ALLOW_SNAPSHOT_ISOLATION database option controls
whether or not the SNAPSHOT isolation level can be used for transactions.

QUESTION 54
You are a database developer on an instance of SQL Server 2008.
Your **Prod** database contains an **InvTrxMaster** table that is partitioned on **WarehouseID**.
Currently, there are four partitions defined, but you need to create an additional partition.
Which action should you take?

A. Consolidate the partitions using the MERGE RANGE clause and re-partition the table.
B. Alter the partition scheme and rebuild the table partitions.
C. Alter the partition function including the SPLIT RANGE clause.
D. Create a temporary table to store the data. Drop and re-create the table.

Answer: C
Section: (none)

Explanation/Reference:
You can use the ALTER PARTITION FUNCTION statement to add a
partition to an existing partitioned table, or to combine two partitions in a partitioned table. The complete syntax for the ALTER PARTITION FUNCTION statement is as follows:
```
ALTER PARTITION FUNCTION partition_function_name(){SPLIT RANGE(boundary_value) |
MERGE RANGE (boundary_value)};
```
The MERGE RANGE clause combines two partitions, and the SPLIT RANGE clause splits a partition into two partitions. Each of these clauses specifies the boundary value for which the split or merge should occur. In addition, before you split partitions, you must ensure that the partition scheme has enough filegroups to

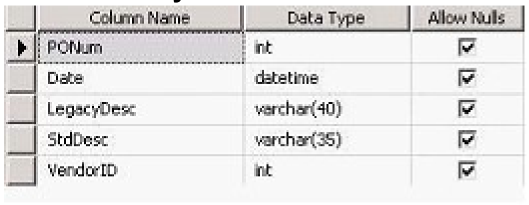
accommodate the new partition, or an error will be generated. You must also specify a value that is different from any other defined boundary value.
You should not consolidate the partitions using the `MERGE RANGE` clause and re-partition the table. The `MERGE RANGE` clause is used when combining two partitions.
You should not alter the partition scheme and rebuild the table partitions. The partition scheme maps partitions to file groups. Modifying the partition scheme would not affect the number of partitions.
You should not create a temporary table to store the data, and then drop and re-create the table. There is no need to re-create the table. This would make the table unavailable to users and would not be the best choice.

QUESTION 55
You need to create a Service Broker solution. Which object should you create first?

A. Contract
B. Dialog
C. Message Type
D. Services

Answer: C
Section: (none)

Explanation/Reference:
The basic steps involved in creating any Service Broker application include:
1. Defining message types
2. Defining contracts
3. Creating queues
4. Creating services
5. Sending and receiving messages

QUESTION 56
You are a database developer on an instance of SQL Server 2008. Your **Purchasing** database contains a table, **POHistory**, which contains data imported from a legacy system.
The **POHistory** table is defined as follows:


Column Name	Data Type	Allow Nulls
PONum	int	✓
Date	datetime	✓
LegacyDesc	varchar(40)	✓
StdDesc	varchar(35)	✓
VendorID	int	✓

Some of the data that was imported into the **LegacyDesc** column needs to be parsed and reformatted. You want to create a function that will accept a string parameter, significantly manipulate the string, and return the manipulated string as output.
You would then like to use this function in a computed column expression in the **StdDesc** column. The **StdDesc** column currently has an index defined on it.
Which action should you take?

A. Create a CLR scalar UDF and set its permission set value to `EXTERNAL_ACCESS`.
B. Create a scalar Transact-SQL function that is recursive and reference it in the computed column.
C. Create a Transact-SQL inline table-valued function and ensure the function is deterministic.
D. Create a CLR scalar UDF and ensure that the function is deterministic and precise.

Answer: D
Section: (none)

Explanation/Reference:
A CLR function is created in a .NET Framework language and then registered in the database so it can be accessed using Transact-SQL. You should consider using CLR functions or procedures when you need to perform CPU-intensive processing, or if the desired functionality can be implemented more efficiently using .NET Framework classes. The .NET Framework offers more robust string manipulation functionality than SQL Server. In this scenario, the function needs to accept a string, perform complex manipulation of the string, and return another manipulated string. Therefore, a CLR scalar function should be used.
In this scenario, you want to reference the function in a computed column of the **POHistory** table, and the column is indexed. You can only have an index on this computed column if the function is deterministic and precise. A function is considered deterministic if it always returns the same values with the same inputs, and precise if it does not perform floating-point calculations. After the CLR scalar UDF is created and compiled, you can use the `CREATE ASSEMBLY` statement to register the compiled assembly in the database and the `CREATE FUNCTION` statement to make the function available from Transact-SQL. Then, you can create a computed column that references the CLR scalar UDF.
You should not create a Transact-SQL inline table-valued function and ensure the function is deterministic. In this scenario, the function only needs to return a single value. Therefore, a table-valued function is not required.
You should not create a scalar Transact-SQL function that is recursive and reference it in the computed column. A recursive function is a function that calls itself, and this is not required in this scenario. Recursive functions should be avoided when possible because they can adversely affect performance.
You should not create a CLR scalar UDF and set its permission set value to `EXTERNAL_ACCESS`. In this scenario, the function does not require the `EXTERNAL_ACCESS` permission set. The `EXTERNAL_ACCESS` permission set provides the function to access the file system, registry, environment variables, and unmanaged code. None of these was required in this scenario. Therefore, the default permission set value of `SAFE` is sufficient.

QUESTION 57
You are a database developer on an instance of SQL Server 2008. The development team has recently developed and compiled an assembly using a .NET language. You register the assembly. You want to be able to reference the appropriate class methods from Transact-SQL. Which action should you take?

A. Create a CLR stored procedure including the `WITH RECOMPILE` option.
B. Create a CLR stored procedure including the `EXTERNAL NAME` clause.
C. Create a CLR user-defined function to call the method.
D. No additional action should be taken other than referencing the methods in the assembly from Transact-SQL.

Answer: B
Section: (none)

Explanation/Reference:
A CLR stored procedure is a stored procedure written in a .NET Framework language. To create a CLR stored procedure, you define a static class method within the .NET language. Then, you compile the class, register the assembly in SQL Server with the `CREATE ASSEMBLY` statement, and use the `CREATE PROCEDURE` statement in SQL Server to reference the assembly. The `EXTERNAL NAME` clause must be specified in the `CREATE PROCEDURE` statement to reference the appropriate method.
You should not create a CLR stored procedure including the `WITH RECOMPILE` option. The `WITH RECOMPILE` clause is used with the `CREATE PROCEDURE` statement to specify that a stored procedure should be recompiled each time it is called.
You should not create a CLR user-defined function to call the method. There is no need to create a CLR user-defined function to make a CLR stored procedure available. After registering the assembly, you can simply create the procedure.
The option that states no additional action should be taken other than referencing the methods in the

assembly from Transact-SQL is incorrect. You must create a stored procedure specifying an `EXTERNAL NAME` clause that names the method(s) that may be referenced.

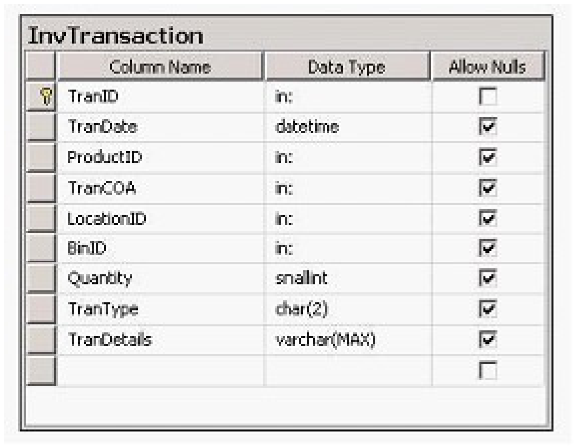
QUESTION 58
You are the designer of a SQL database on an instance of SQL Server 2008. You have an application that uses image files that you receive from
business partners. Because these files can be large, you want them physically stored on the file system. You also want to ensure that these files are
backed up when the database is backed up.
What should you do?

A. Create a CLR user-defined type.
B. Define a **hierarchyid** data type column.
C. Define a column that implements FILESTREAM storage.
D. Define an **image** data type column and create a trigger to back up the files when a backup is taken.

Answer: C
Section: (none)

Explanation/Reference:
FILESTREAM storage is implemented to store large binary objects, such as
image or video files, as files on the file system and be able manage to them using Transact-SQL. FILESTREAM data can also be accessed using
Win32 APIs. FILESTREAM storage is tightly integrated with most database functionality, including backup and recovery. When you take a database
backup, FILESTREAM storage is also backed up unless you override this functionality by performing a partial backup. To create a table that can
store FILESTREAM data, you create a table that contains a column of the **varbinary(max)** data type and include the `FILESTREAM` attribute. For
example, you could use the following Transact-SQL statement to create a table with a **Blueprint** column that stores FILESTREAM data:
```
CREATE TABLE MyTable(
ID uniqueidentifier ROWGUIDCOL NOT NULL UNIQUE,
Blueprint varbinary(max) FILESTREAM NULL);
```
You should also note that to use FILESTREAM data, you must first use SQL Server Configuration Manager or the **sp_configure** system stored
procedure to enable FILESTREAM, because it is not enabled by default.
You should not create a CLR user-defined type. A CLR user-defined type is a user-defined type that is created using a .NET Framework language.
CLR user-defined types are used to implement more complex data types that cannot be implemented using traditional methods.
You should not define a **hierarchyid** data type column. A **hierarchyid** data type is a special variable-length, CLR-supplied data type that can be
used to represent hierarchical data. When you define a **hierarchyid** data type column, you can use various system-supplied methods, such as
GetRoot, **GetDescendant**, **GetLevel**, **Read**, and **Write**, which allow you to perform tasks on the hierarchy. Each value stored in the table is an
internal binary value that represents the row's position in the hierarchy. Using a **hierarchyid** data type simplifies management of data that is
represented in a tree-like structure.
You should not define an **image** data type column and create a trigger to backup the files when a backup is taken. An image column's contents are
stored within the database, rather than external to the database on the file system.

QUESTION 59
You are a database developer on an instance of SQL Server 2008. Your **Prod** database contains an **InvTransaction** table defined as follows:



InvTransaction

	Column Name	Data Type	Allow Nulls
🔑	TranID	int	☐
	TranDate	datetime	☑
	ProductID	int	☑
	TranCOA	int	☑
	LocationID	int	☑
	BinID	int	☑
	Quantity	smallint	☑
	TranType	char(2)	☑
	TranDetails	varchar(MAX)	☑
			☐

Users frequently run the following query:

```
SELECT TranDate, TranType, QuantityFROM InvTransaction WHERE TranType = @type
ANDTranDate BETWEEN @begin AND @end ANDQuantity > 100;
```

Which action should you take to improve the query's performance?

A. Create a clustered composite index on **TranDate**, **TranType**, and **Quantity**.
B. Create a nonclustered index on **TranDate**.
C. Create a clustered index on **TranDate** and a composite non-clustered index on the **TranType** and **Quantity** columns.
D. Create a nonclustered index on **TranDate** and include **TranType** and **Quantity**.

Answer: D
Section: (none)

Explanation/Reference:
An index created on all the columns in the `SELECT` list increases the performance of the query because the data required for the query can be directly retrieved from the index. Therefore, only the index pages containing the index will be scanned to return the data, improving query performance. An index that covers all the columns in the `SELECT` list is referred to as a covering index.
You should not create a clustered composite index on **TranDate**, **TranType**, and **Quantity**, or create a clustered index on **TranDate** and a composite non-clustered index on **TranType** and **Quantity**. Only one clustered index is allowed for each table. A clustered index denotes the physical order of the table. In this scenario, the **InvTransaction** table contains a primary key. When a primary key is created, a clustered index is created automatically if one does not already exist.
You should not create a nonclustered index on **TranDate**. This might improve query performance somewhat, but creating a covering index would provide better query performance.

QUESTION 60
You are a database developer. You plan to design a database solution by using SQL Server 2008. The database supports a warehousing application and contains data from two warehouses in a table named Product. The Product table contains a warehouse indicator field named warehouse_id. The two

Warehouse B contains 55,000 items.
The solution uses a third-party application that runs on SQL Server 2008. The application uses a stored procedure that returns the warehouse inventory based on the warehouse_id parameter. Users report that occasionally, the system performance is unacceptable when the Warehouse A inventory is queried.
You cannot modify the stored procedures in the application. You need to ensure acceptable performance

when users query the inventory of Warehouse A.
What should you do?

A. Create a clustered index on the warehouse_id column.
B. Create a nonclustered index on the warehouse_id column.
C. Create a plan guide that sets the MAXDOP query hint to 1.
D. Create a plan guide that uses the OPTIMIZE FOR clause for Warehouse A.

Answer: D
Section: (none)

Explanation/Reference:
były również odpowiedzi a

QUESTION 61
You are a database developer. You plan to design a database solution by using SQL Server 2008. The database contains a large table that is infrequently updated. Users execute a query against the table. The query requires the execution of a complex calculation that involves multiple columns for a given row. You discover that the query performance is poor because the query is CPU intensive.
You need to reduce the effect of this query on the server.
What should you do?

A. Create a computed column on the table.
B. Create a persisted computed column on the table.
C. Create an index on each field used by the calculation.
D. Create a view on the table that includes the calculation.

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 62
You have a SQL Server Integration Services (SSIS) package that contains an Execute Process task. You need to schedule the SSIS package to run on a regular basis. What should you do?

A. Create a credential and a login. Configure a SQL Server Agent job to run the package by using the login.
B. Create a credential and a proxy. Configure a SQL Server Agent job to run the package by using the proxy.
C. Create a login and map a user to the login. Add the user to the db_owner role.Configure a SQL Server Agent job to run the package by using the login.
D. Create a login and map the user to a login. Add the user to the db_securityadmin role.Configure a SQL Server Agent job to run the package by using the login.

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 63

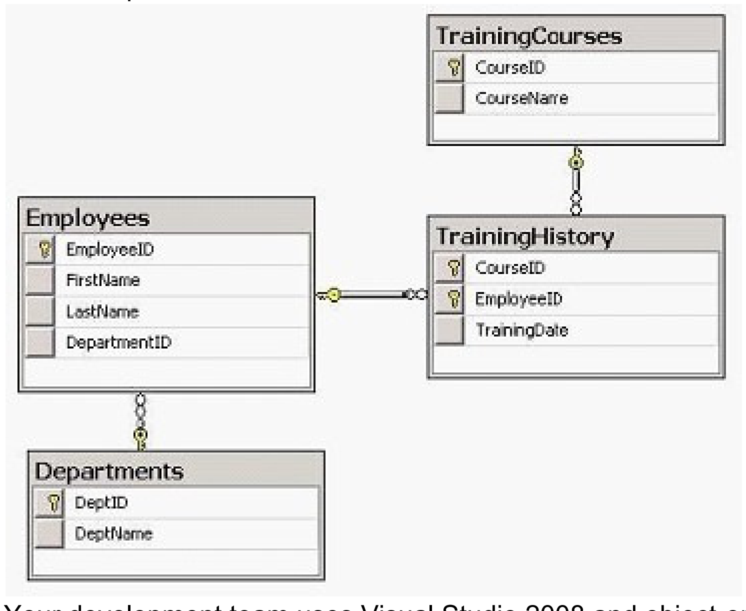
You are a database developer. You plan to design a database solution by using SQL Server 2008. Account managers in your company store order data in a database. Your company requires a list of customers for each account manager. The list must be sorted in the descending order of the order amount.
You create a query that generates the list at the end of each month. You need to ensure that the query executes as quickly as possible.
What should you do?

A. Create a cursor that returns each account manager, and then sort the order data by order amount.
B. Use a SELECT statement that uses the OVER clause to rank the customers by order amount for each account manager.
C. Create a correlated subquery to return the order amount for each account manager. Sort the results first by account manager and then by order amount.
D. Create a table-valued function that returns the order amount for a specific account manager, and then create a query by using the CROSS APPLY clause to list each account manager. Sort the results first by account manager and then by order amount.

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 64
You maintain a database on an instance of SQL Server 2008. Your database contains the following tables and relationships:



Your development team uses Visual Studio 2008 and object-oriented programming (OOP) techniques to create applications that access data in your database. You want to reduce the coding complexity for developers writing data access code, and minimize application changes required when the database schema changes.
You want developers to be able to instantiate **Employee** and **Course** objects and access column data using object properties.

What should you do?

A. Create a data model using the Entity Framework that includes an entity for each database table and an association for each `FOREIGN KEY`
constraint.
B. Create a data model using the Entity Framework that includes the **Employee** and **Course** entities.
C. Create application-specific views and use `INSTEAD OF` triggers to perform any update or delete operations.
D. Create stored procedures that developers must use for data access.

Answer: B
Section: (none)

Explanation/Reference:
The Entity Framework allows you to create data models that allow developers to query a conceptual model without being concerned with the underlying database schema. An Entity Data Model (EDM) abstracts the conceptual model from the physical database schema. An EDM contains a conceptual model, a logical storage model, and mappings between the two models, which are each stored in special XML files. The conceptual model is stored in a conceptual schema definition language (**.csdl**) file. The storage model is stored in a storage schema definition language (**.ssdl**) file. The mappings are stored in a mapping specification language (**.msl**) file. The EDM consists of entities and relationships. Entities can be exposed as objects, so they can have properties and be instantiated in application code. Relationships can be defined that relate entities in the conceptual model and support inheritance between entities.
Creating entities in the EDM that represent logical business entities simplifies the conceptual view of the underlying table structures. For example, in this scenario, you would have an **Employee** entity and a **Course** entity. An **Employee** object might have a **Department** property that is the employee's department, and a **Courses** property that is a collection containing the courses that the employee has previously taken. You would also have a **Course** entity. A **Course** object might have a collection of attendees as one of its properties. The Entity Framework uses the XML files representing the model to translate the operations performed on the data model to operations against the respective tables in the underlying data source. If the database schema changes, the **.ssdl** and **.msl** files can be changed to reflect changes in the data model without requiring changes to either the conceptual model or the application code. The Entity Framework provides a special query language called Entity SQL. Entity SQL is similar to SQL, but has different syntax. Developers may use Entity SQL to write data access code against the conceptual model.
You should not create a data model using the Entity Framework that includes an entity for each database table and an association for each `FOREIGN KEY` constraint. An EDM is intended to abstract the conceptual model from the underlying database structure.
You should not create application-specific views and use `INSTEAD OF` triggers to perform update or delete operations, or create stored procedures that developers must use for data access. Views, `INSTEAD OF` triggers, and stored procedures are often used to increase performance, prevent SQL injection attacks, or centralize or hide SQL code. However, database schema changes may still require that these constructs and application code be modified. In this scenario, you wanted to minimize the impact if the database schema changed.

QUESTION 65
You are a database administrator on an instance of SQL Server 2008. Your development team is using the **Test** database. Each developer has a SQL login and is a member of the **db_owner** fixed database role. By default, all developers use the **dbo** schema.
Your company recently hired a contract development team to work on a special development project. The contractors need the ability to create new database tables, views, and stored procedures in the **Test** database.
You also want to meet the following requirements:
Database objects created by the contractors must be separately maintained and easily differentiated from the objects created by the development team.
Contractors must be able to query and view the definitions of all existing database objects, but not alter them.
The administrative effort required to manage and secure database objects should be minimized.

Which actions should you take?

A. Create a database schema named **Special**.
   Create a **Contractor** database role and grant the role `SELECT` and `VIEW DEFINITION` permissions on the **dbo** schema and the `CONTROL` permission on the **Special** schema.
   Grant the desired `CREATE` permissions to the **Contractor** role.
   Create a user account for each contractor with the **Special** schema as the default schema.
   Make each contractor a member of the **Contractor** role.
B. Create a database schema named **Special**.
   Grant the necessary permissions to the **Special** schema.
C. Create a database role named **Contractor** and grant the role `SELECT` and `VIEW DEFINITION` permissions on the **dbo** schema and
   the `CONTROL` permission on the **Special** schema.
   Create a user account for each contractor.
   Make each contractor a member of the **Contractor** role.
D. Create a DDL trigger that fires each time a contractor creates an object.
   Code the trigger to assign the appropriate object permissions.

Answer: A
Section: (none)

Explanation/Reference:
You should perform the following actions:
Create a database schema named **Special**.
Create a **Contractor** database role and grant the role `SELECT` and `VIEW DEFINITION` permissions on the **dbo** schema and the `CONTROL` permission on the **Special** schema.
Grant the desired `CREATE` permissions to the **Contractor** role.
Create a user account for each contractor with the **Special** schema as the default schema.
Make each contractor a member of the **Contractor** role.

Schemas contain database objects. Using schemas allows you to manage ownership and permissions of database objects more effectively. In this scenario, you should create a separate schema to contain all database objects that contractors create. This will allow you to keep the contractors' database objects logically separated from the objects created by the development team. The schema name is displayed in Object Manager to make objects easier to identify. You can also grant permissions at the schema level to simplify the management of permissions. If you grant a permission at the schema level, the same permission is implicitly granted for all database objects within the schema, even future objects. You can use the `CREATE SCHEMA` statement to create a schema, and optionally specify an `AUTHORIZATION` clause to specify the schema's owner. The schema's owner may be a user or role. If no schema owner is specified, **dbo** is the default schema owner. In this scenario, you could create the **Special** schema owned by **dbo** using the following statement:

```
CREATE SCHEMA Special;
```

Next, you need to give contractors the ability to perform their necessary tasks. The best way to implement this is to create a database role and grant the role the needed permissions. Then, you can make each contractor a member of the role. Permissions may be granted at the schema level instead of at the object level. Therefore, you could use the following Transact-SQL to create the **Contractor** role and grant the role the necessary permissions:

```
-- Create the Contractor role CREATE ROLE Contractor;GO
-- Allows contractors to query and view the definitions of table in -- the dbo
schema GRANT SELECT, VIEW DEFINITIONON SCHEMA::[dbo] to [Contractor];GO
-- Allows contractors to control Special schemaGRANT CONTROL ON SCHEMA::[Special]
to [Contractor];GO
-- Allows contractors to create tables, views, and stored procedures-- in the
database GRANT CREATE TABLE, CREATE VIEW, CREATE PROCEDURE TO [Contractor];
GO
```

After you create the **Contractor** role, you should then create contractor accounts with the **Special** schema as a default schema and make them a member of the **Contractor** role. When a member of the role creates a database object, it will be created in the user's default schema. The following Transact-SQL will create the

Contractor1 user with a default schema of **Special** and assign the user to the **Contractor** role:

```
CREATE LOGIN [Contractor1] WITH PASSWORD=N'12345'GO
CREATE USER [Contractor1] FOR LOGIN [Contractor1] WITH DEFAULT_SCHEMA=[Special]GO

EXEC sp_addrolemember N'Contractor ', N'Contractor1'
--Allow user to connect to database GRANT CONNECT TO [Contractor1]
```

You can specify a schema as the user's default schema before the schema has even been created. You should note that if the user is authenticated via a Windows group, the user will have no default schema assigned. If such a user creates an object, SQL Server will create a new schema in which the object is created. The new schema created will have the same name as the user that created the object.
When a schema is no longer needed, such as when the project is complete, you can drop it using the `DROP SCHEMA` statement. To be dropped, a
schema must be empty, or the `DROP SCHEMA` statement will generate an error. Therefore, in this scenario, you could first move the objects from the **Special** to the **dbo** schema. You can use the `TRANSFER` clause of the `ALTER SCHEMA` statement to transfer objects from one schema to another. After transferring all objects to another schema, you could drop the schema. The following Transact-SQL will move the **ContractorTbl** table from the **Special** schema to the **dbo** schema and then drop the **Special** schema:

```
ALTER SCHEMA dbo TRANSFER Special.ContractorTbl;DROP SCHEMA Special;
```

All of the other options are incorrect because they would not meet the requirements in this scenario

QUESTION 66
You are a database developer on an instance of SQL Server 2008. You have an **Employee** table created with the following statement:

```
CREATE TABLE Employee (
EmpID int IDENTITY(1,1) PRIMARY KEY,
LastName varchar(35) NOT NULL,
FirstName varchar(25) NOT NULL,
HireDate datetime,
Salary money,
MgrID int);
```

The **MgrID** column will contain the **EmpID** associated with each employee's manager. You want to create a self-referencing relationship to support
this.
Which action should you take?

A. Create a DML trigger.
B. Create a `FOREIGN KEY` constraint on the **EmpID** column.
C. Create a `FOREIGN KEY` constraint on the **MgrID** column.
D. Create a `CHECK` constraint on the **MgrID** column.

Answer: C
Section: (none)

Explanation/Reference:
A `FOREIGN KEY` constraint is used to establish a relationship between two
tables or to establish a self-referencing relationship within a single table. You can create a `FOREIGN KEY CONSTRAINT` using the `ALTER TABLE`
statement. The `REFERENCES` clause specifies the referenced column, which in this scenario is the **EmpID** column. This relates each employee with
the row in the **Employee** table that identifies the employee's manager. The `FOREIGN KEY` clause can be omitted and use only the `REFERENCES`
clause when creating a `FOREIGN KEY` constraint. You can also include `ON DELETE` and `ON UPDATE` clauses when creating the `FOREIGN KEY`
constraint to control how referential integrity is enforced.
A `CHECK` constraint consists of a Boolean expression that evaluates to either `TRUE`

or `FALSE`. If the expression evaluates to `TRUE`, the value is allowed for the column, and if the expression evaluates to `FALSE`, the value is not
allowed for the column. `CHECK` constraints can be defined at the table level or column level, but only `CHECK` constraints defined at the table level
can use columns other than the constrained column in the constraint expression.
DML triggers contain Transact-SQL code that executes when a DML operation, such as an `INSERT`, `UPDATE`, `MERGE`, or `DELETE`, is performed. `INSTEAD OF` triggers fire in place of the triggering operation, and `AFTER` triggers fire after the triggering operation
completes successfully. You might use a DML trigger to prevent the original triggering operation or to perform additional or alternative actions.

QUESTION 67
You have a table in a database that contains 30 million rows. You need to ensure that the table meets the following requirements:

- Enables queries on all string values within the table
- Enables queries to be returned by order of proximity
- Minimizes the amount of time required to execute queries

What should you do?

A. Create a filtered index.
B. Create a clustered index.
C. Configure Service Broker.
D. Configure a Full-Text-Search.

Answer: D
Section: (none)

Explanation/Reference:
SQL Server 2008 provides the functionality for applications and users to issue full-text queries against character-based data in SQL Server tables. Before full-text queries can be run on a given table, the database administrator must create a full-text index on the table. The full-text index includes one or more character-based columns in the table. These columns can have any of the following data types: char, varchar, nchar, nvarchar, text, ntext, image, xml, varbinary, or varbinary(max). Each full-text index indexes one or more columns from the base table, and each column can have a specific language.

QUESTION 68
You are a database developer on an instance of SQL Server 2008. You have a large production database that contains medical record data for multiple medical practices.
The **PatientVisit** table contains over two million rows of data about patient visits. The **PatientVisit** table is updated each day for new patient visits, but information for past patient visits is rarely modified. You have several queries that use the information in the **PatientVisit** table, but most queries are performing poorly.
What should you do?

A. Create a filtered index on the **PatientVisit** table.
B. Partition the **PatientVisit** table by date.
C. Include the `INDEX(0)` query hint with all queries on the **PatientVisit** table.
D. Create a horizontally partitioned view on the **PatientVisit** table.

Answer: B
Section: (none)

Explanation/Reference:

Partitioning makes large tables easier to manage. Partitioning also speeds up access because you can access smaller subsets of data more quickly and maintain the data integrity. In addition, you can spread these partitions across file groups to improve I/O performance.
You should not create a filtered index. A filtered index is a nonclustered index that is defined including a specific `WHERE` clause to optimize the index for specific queries.
You should not include the `INDEX(0)` query hint with all queries on the **PatientVisit** table. If a clustered index exists on the table, the `INDEX(0)` hint forces a clustered index scan on the table. If no clustered index exists, the hint forces a full table scan. You can use the `INDEX` table hint to improve performance, but the index that you would need to use to optimize performance would depend on the actual query. It would not be appropriate to include the `INDEX(0)` hint for all queries.
You should not create a horizontally partitioned view on the **PatientVisit** table. Partitioned views are used when you have similar data stored in multiple tables and want to create a view to allow access to all of the data as if it were stored in a single table. Partitioned views are implemented by creating a view that queries several tables and combines the results using the `UNION ALL` operator. A horizontally partitioned view is a partitioned view that spreads across multiple servers. Each server contains a portion of the data, but the view created makes it appear as if all of the data is in a single table. A horizontally partitioned view can improve performance and increase availability, but would not be applicable in this scenario.

QUESTION 69
You are a database developer on an instance of SQL Server.
Your **Prod** database currently contains a view named **OrderDetailsView** that joins five tables.
Users will frequently perform queries using the view, and you want to improve the performance of these queries.
Which action should you take?

A. Create a filtered index on the view.
B. Partition the view.
C. Create a full-table nonclustered index on the view.
D. Create a unique clustered index on the view.

Answer: D
Section: (none)

Explanation/Reference:
Creating a unique clustered index on a view increases the view's performance. Views with clustered indexes are referred to as indexed views. In an indexed view, the result set is stored in the database, similar to storing a table with a clustered index. Because a result set does not have to be generated each time the view is referenced, it can drastically improve performance. To create an indexed view, the view must use schema binding. Therefore, in this scenario, you might need to alter the view to include the `WITH SCHEMABINDING` clause for the index to be created successfully on the view. After you create an indexed view, the optimizer will use the view index even for queries that do not directly reference the view in the `FROM` clause.
You should not create a full-table nonclustered index on the view. To index a view, you create a unique clustered index on the view, not a nonclustered index.
You should not partition the view. Partitioned views are used when you have similar data stored in multiple tables and want to create a view to allow access to all of the data as if it were stored in a single table. Partitioned views are implemented by creating a view that queries several tables and combines the results using the `UNION ALL` operator. Although in some situations a partitioned view is helpful, one would not be applicable in this scenario.
You should not create a filtered index on the view. A filtered index is a nonclustered index that is defined including a specific `WHERE` clause to
optimize the index for specific queries. The index uses the `WHERE` clause condition to index only specific rows in the table. Using a filtered index can improve performance in many situations and can reduce the space required for the index. However, a filtered index cannot be created on a view.

QUESTION 70
You are a database developer. You plan to design a database solution by using SQL Server 2008. The database contains two tables named Supplier and Product. There is a foreign key constraint between the Supplier and Product tables on the SupplierID column. The Supplier table contains a row that has the SupplierID value as 0. The 0 value indicates that the supplier is deleted. Certain transactions delete the supplier records from the Supplier table. You need to ensure that if a supplier is deleted, the SupplierID value in the Product table is set to 0.
What should you do?

A. Create a FOR DELETE trigger on the Product table that updates the SupplierID value to 0 in the Products table for the deleted supplier.
B. Create a FOR DELETE trigger on the Supplier table that updates the SupplierID value to 0 in the Products table for the deleted supplier.
C. Create a default constraint on the SupplierID column in the Product table that sets the value to 0. Set the ON DELETE property of the foreign key constraint to NULL.
D. Create a default constraint on the SupplierID column in the Product table that sets the value to 0. Set the ON DELETE property of the foreign key constraint to Default.

Answer: D
Section: (none)

Explanation/Reference:

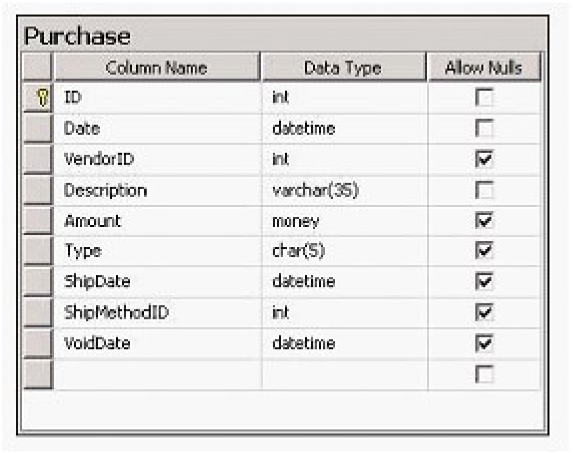
QUESTION 71
You have a database that contains two tables named Table1 and Table1_Details. Table1_Details contains details about items in Table1. You need to ensure that when an item is removed from Table1, all related items are removed from Table1_Details. You must achieve this goal by using the minimum amount of Transact-SQL code. What should you do?

A. Create a foreign key relationship. Set Cascade Delete to Null.
B. Create a foreign key relationship. Set Cascade Delete to True.
C. Create a trigger on Table1_Details that fires on the Delete action.
D. Create a stored procedure that deletes all related items from Table1_Details.

Answer: B
Section: (none)

Explanation/Reference:
ON DELETE CASCADE
Specifies that if an attempt is made to delete a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are also deleted. If cascading referential actions have also been defined on the target tables, the specified cascading actions are also taken for the rows deleted from those tables.

QUESTION 72
You are a database developer on an instance SQL Server 2008. You have a **Purchase** table defined as follows:



Purchase

Column Name	Data Type	Allow Nulls
ID	int	☐
Date	datetime	☐
VendorID	int	✓
Description	varchar(35)	☐
Amount	money	✓
Type	char(5)	✓
ShipDate	datetime	✓
ShipMethodID	int	✓
VoidDate	datetime	✓
		☐

The **Purchase** table contains millions of rows and you want to optimize query performance. Most purchases have not been voided, and do not contain a **VoidDate**. You are creating a stored procedure that will query the **Purchase** table based on a specified date range and return information for all non-voided orders, and you want the stored procedure's query to perform optimally.
Which action should you take?

A. Create a full-table nonclustered index on the **VoidDate** column.

B. Create an indexed view on the **Purchase** table and use the view in the stored procedure.

C. Create a partitioned view and use the partitioned view in the stored procedure.

D. Create a filtered index that uses the **VoidDate** column in the `WHERE` clause.

Answer: D
Section: (none)

Explanation/Reference:
A filtered index is a nonclustered index that is defined including a specific `WHERE` clause to optimize the index for specific queries. The index uses the `WHERE` clause condition to index only specific rows in the table. Using a filtered index can improve performance in many situations and can reduce the space required for the index.
In this scenario, most of the rows of the **Purchase** table contained a `NULL` value for the **VoidDate** column, but the stored procedure retrieves only non-voided orders. Therefore, you should create a filtered index that only includes entries for non-voided orders. You could use the following statement to create the filtered index:

```
CREATE NONCLUSTERED INDEX FI_PurchaseNonVoided ON Purchase (Date)WHERE VoidDate
IS NOT NULL;
```

In addition to creating a filtered index that does not include rows with `NULL` value, you can also use filtered indexes to restrict the indexed rows based on a complex `WHERE` clause condition. For example, you might use the following `WHERE` clause when creating a filtered index to index only specific types of purchases:

```
WHERE Type IN ('PO', 'ONL') AND ShipDate > '20081231'
```

You should not create a full-table nonclustered index on the **VoidDate** column. In this scenario, a filtered index would offer better performance because most of the rows in the **Purchase** table do not contain a void date.
You should not create an indexed view on the **Purchase** table and use the view in the stored procedure. An indexed view is a view that has a unique clustered index defined on it. In an indexed view, the result set is stored in the database, similar to storing a table with a clustered index. Because a result set does not have to be generated each time the view is referenced, this can drastically improve performance. However, in this scenario, you should use a filtered index.
You should not create a partitioned view and use the partitioned view in the stored procedure. Partitioned views are used when you have similar data stored in multiple tables and want to create a view to allow access to all of the data as if it were stored in a single table. Partitioned views are implemented by creating a view that queries several tables and combines the results using the `UNION ALL` operator. Although in some situations a partitioned view is helpful, using one would not be applicable in this scenario.

QUESTION 73
You are a database administrator on an instance of SQL Server 2008. You have implemented database roles and used role membership to grant all user permissions.
Your company has recently hired a contractor who needs permissions associated with the**Developer** role. The **Developer** role has permission to query all tables, as well as to create database objects. Due to privacy concerns, you do not want the contractor to be able to access specific table columns in the **Employee** table, which the **Developer** role allows.
You create the **Contractor** user. You want to assign the appropriate permissions with the least administrative effort.
Which additional action(s) should you take?

A. Create a new database role with only the allowed contractor permissions and make the**Contractor** user a member of the new role.
B. Create an application role for the **Contractor** user.
C. Make the **Contractor** a member of the **Developer** role and explicitly deny permissions at the column level.
D. Explicitly grant the allowed contractor permissions to the **Contractor** user.

Answer: C
Section: (none)

Explanation/Reference:
You should not create a new database role with only the allowed contractor permissions and make the **Contractor** user a member of the new role. In this scenario, you should grant membership to the role, but override permissions at the user level. Creating a separate role would require more effort than granting membership to the role, but overriding permissions at the user level. If you had a team of contractors that all needed similar access or the desired permissions were significantly different than those provided with the **Developer** role, then you might create a new role to use for all contractors.
You should not create an application role for the **Contractor** user. Application roles are not granted specifically to users. Applications roles are activated by an application and are used to restrict application users to only the actions allowed for the application.
You should not explicitly grant the allowed contractor permissions to the **Contractor** user. This would require much more effort because all permissions granted to the **Developer** role would have to be explicitly granted.

QUESTION 74
You are designing a maintenance strategy for a database that contains several views. The views will be assigned custom permissions. You need to recommend a solution that will allow developers to modify the views without affecting the view's existing permissions. What should you recommend?

A. Create a new view.
B. Alter the existing view.
C. Rename the existing view.
D. Drop the existing view and then recreate the view.

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 75

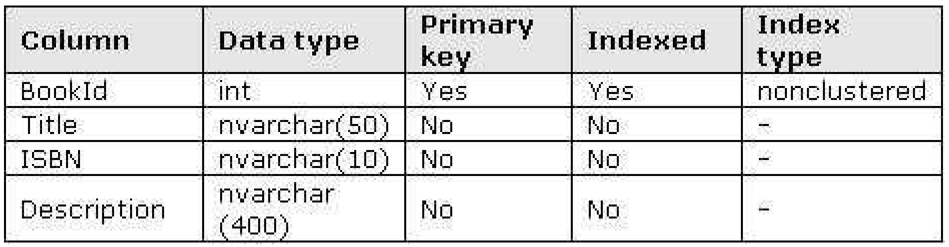
You are a database developer. You plan to design a database solution by using SQL Server 2008. The database will contain information on retail sales transactions of more than 500 stores. The marketing department uses the solution to analyze daily sales patterns for each store. Users report that the solution takes a long time to retrieve the required data. You need to ensure that the solution provides results in the minimum possible time.
What should you do?

A. Create a nonclustered index on a view of the sales transactions.
B. Create a covering index on a view that aggregates the sales transactions.
C. Create a clustered index on a view that aggregates the sales transactions.
D. Create a nonclustered index on a view that aggregates the sales transactions.

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 76
You have a table named Books that contains information about books. Books has the columns in the following table.



Column	Data type	Primary key	Indexed	Index type
BookId	int	Yes	Yes	nonclustered
Title	nvarchar(50)	No	No	-
ISBN	nvarchar(10)	No	No	-
Description	nvarchar (400)	No	No	-

You plan to create several queries that will filter on Title and ISBN. The queries will return values from Title, ISBN, and Description.
You need to recommend an indexing solution to meet the following requirements:
Minimize the amount of time required to return the results of the planned queries Minimize the number of indexes
What should you recommend?

A. Create a nonclustered index on each column.
B. Create a clustered index on Title, ISBN and Description as the key value.
C. Create a clustered index on Title and ISBN and set the index fill factor to 75.
D. Create a nonclustered index on Title and ISBN and include the Description column.

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 77
You are a database developer. You plan to design a database solution by using SQL Server 2008. The database application has a table named Transactions that contains millions of rows. The table has multiple columns that include transaction_id and transaction_date. There is a clustered index on the transaction_id

column. There is a nonclustered index on the transaction_date column. You discover that the following query takes a long time to execute. SELECT transaction_id, transaction_date, transaction_notes FROM transactions WHERE transaction_type_id = 'FXO'
AND transaction_date between @start_date and @end_date The summary of the execution plan is as shown in the following code segment.
|--Filter(WHERE:([transaction_type_id]='FXO')
|--Nested Loops(Inner Join)
|--Index Seek(OBJECT:([transactions]. [nc_transactions_transaction_date]) |--Clustered Index Seek(OBJECT:([transactions]. [PK_transactions_transaction_id]) You need to ensure that the query retrieves data in minimum possible time.
What should you do?

A. Create a nonclustered index on the transaction_type_id column.
B. Create a nonclustered index on the transaction_date and transaction_type_id columns.
C. Create a nonclustered index on the transaction_date column and include the transaction_type_id and transaction_notes columns.
D. Create a nonclustered index on the transaction_date and transaction_type_id columns and include the transaction_notes column.

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 78
You are the database developer on an instance of SQL Server 2008. You are designing a new database to track projects and project managers within your organization.
You must create a data model representing the relationship between the **Project** entity and the **ProjectManager** entity that will support the following requirements:
Each project may have one or more project managers, depending on the size of the project.
Each project manager may be assigned to manage a single project or multiple projects.
Each project has a specific date on which the project is scheduled to begin.

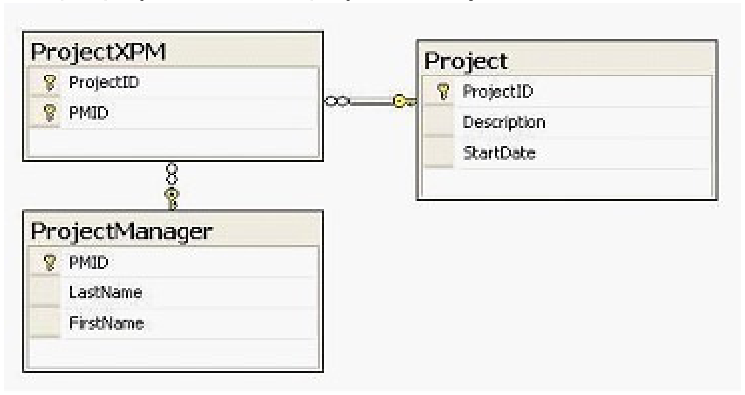
Which action should you take?

A. Create a one-to-one relationship between the **Project** entity and the **ProjectManager** entity.
B. Create a many-to-many relationship between the **Project** entity and the **ProjectManager** entity.
C. Create a one-to-many relationship from the **ProjectManager** entity to the **Project** entity
D. Create a one-to-many relationship from the **Project** entity to the **ProjectManager** entity.

Answer: B
Section: (none)

Explanation/Reference:
In this scenario, each project may have one or more project managers, and each project manager may be assigned to multiple projects. In addition, each project has a defined start date. To implement these requirements, you must use a many-to-many relationship. Many-to-many relationships can be implemented by creating another table, referred to as a junction table, to join the entities. The junction table contains a composite primary key consisting of the primary keys of the joined tables, and `FOREIGN KEY` constraints on each of the primary key columns to reference the original tables. The junction table may also include other applicable columns. For example, in this scenario, you could use the following Transact-SQL statements to create the **Project** and **ProjectManager** tables and a junction table, **ProjectXPM**, to implement the many-to-many relationship:

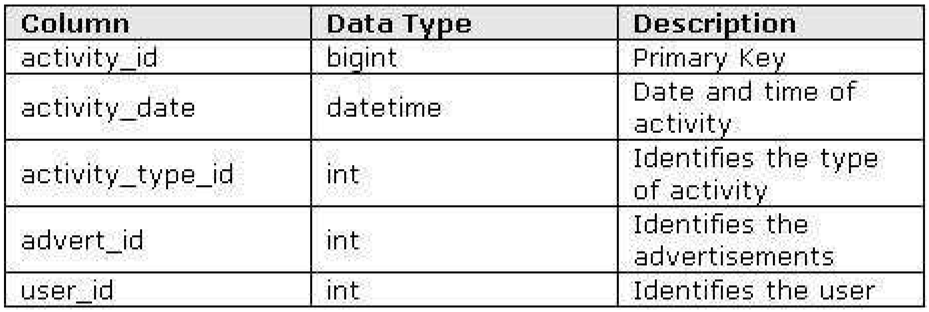
```
CREATE TABLE Project (ProjectID int PRIMARY KEY,Description varchar(25),StartDate
datetime);
CREATE TABLE ProjectManager (PMID int PRIMARY KEY,LastName varchar(30),FirstName
varchar(30));
CREATE TABLE ProjectXPM (ProjectID int,PMID int,CONSTRAINT PK_Project_PMPRIMARY
KEY CLUSTERED (ProjectID, PMID),FOREIGN KEY (ProjectID) REFERENCES Project
(ProjectID),FOREIGN KEY (PMID) REFERENCES ProjectManager (PMID));
```
Each row in the **Project** table represents a single project within the organization and includes the project's start date. Each row in the **ProjectManager** table represents a project manager who may be assigned to projects within the organization. Each row in the **ProjectXPM** table represents a project manager assigned to a specific project. The **ProjectXPM** table contains a composite primary key consisting of the combination of **ProjectID** and **PMID**. This ensures that the table may include multiple project managers for each project and multiple projects for each project manager. The data model in this scenario would resemble the following:



You should not create a one-to-one relationship between the **Project** entity and the **ProjectManager** entity. This data model would allow each project manager to be assigned to only one project and each project to be assigned only one project manager, but would not allow projects to have

multiple project managers or allow project managers to manage multiple projects.
You should not create a one-to-many relationship from the **ProjectManager** entity to the **Project** entity. This data model would allow each project manager to be assigned to multiple projects, but would only allow each project to be assigned a single project manager.
You should not create a one-to-many relationship from the **Project** entity to the **ProjectManager** entity. This data model would allow each project to have one or more project managers, but would allow a project manager to be assigned to only one project.

QUESTION 79
You are a database developer. You plan to design a database solution by using SQL Server 2008. The database supports a Web site and captures user interactions. These interactions are stored in the Activity table of the User_Activity database. Data older than six months is archived to the Activity table of the Archive_Activity database on a different instance of SQL Server 2008. The structure of the Activity table is as shown in the following table.



Column	Data Type	Description
activity_id	bigint	Primary Key
activity_date	datetime	Date and time of activity
activity_type_id	int	Identifies the type of activity
advert_id	int	Identifies the advertisements
user_id	int	Identifies the user

You plan to design a solution that allows a single query to generate a report that summarizes user interactions

for the last 12 months.
You need to ensure that the solution is implemented.
Which two actions should you perform? (Each correct answer presents part of the solution. Choose two.)

A. Create a partition function and a partition scheme.
B. Modify the Activity tables to use the partition scheme.
C. Move the archived data back to the User_Activity database.
D. Create a view by using the UNION ALL clause to retrieve data from the two Activity tables.

E. Create CHECK constraints on the two Activity tables to limit the values in the activity_date column to an exclusive range.

Answer: DE
Section: (none)

Explanation/Reference:

QUESTION 80
You are a database developer on an instance of SQL Server. You have a table named **ProjectDetails** that is partitioned by the **ProjectDate** column.
You have several queries that are performing poorly. To improve performance, you want to create a table-aligned index on the **ProjectDetails** table.

Which action should you take?

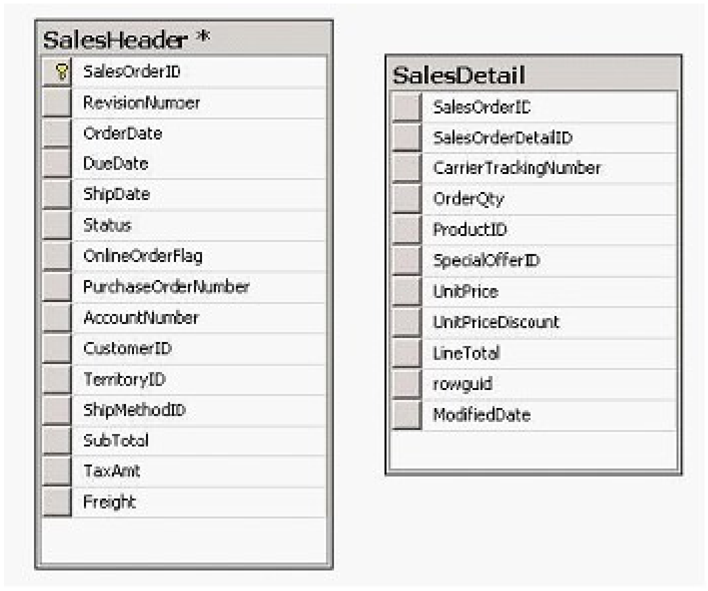
A. Create a partition function on the same data type as the partition function used by the **ProjectDetails** table. Create an index using the new partition function.
B. Create a filtered index on the **ProjectDetails** table.
C. Create an index using the same partition scheme and key as the **ProjectDetails** table.
D. Create an index on the **ProjectDate** column and rebuild the **ProjectDetails** table.

Answer: C
Section: (none)

Explanation/Reference:
To create an index that will be aligned with the table, you must specify the same partition scheme and key that was used when the table was created. For example, in this scenario, if you specified the **ProjectDetails_PS** partition scheme when creating the **ProjectDetails** table, you might use the following statement to create a table-aligned index:
```
CREATE UNIQUE CLUSTERED INDEX IX_ProjectDetailsON ProjectDetails(ProjectDate)ON
ProjectDetails_PS (ProjectDate);
```
You should not create a partition function on the same data type as the partition function used by the **ProjectDetails** table and create an index using the new partition function. There is no need to create an additional partition function.
You should not create a filtered index. A filtered index is a nonclustered index that is defined including a specific `WHERE` clause to optimize the index for specific queries.
You should not create an index on the **ProjectDate** column and rebuild the **ProjectDetails** table. There is no need to rebuild the **ProjectDetails** table. In addition, to create an index that is aligned with the table, you must specify the same partition scheme used with the **ProjectDetails** table when creating the index.

QUESTION 81

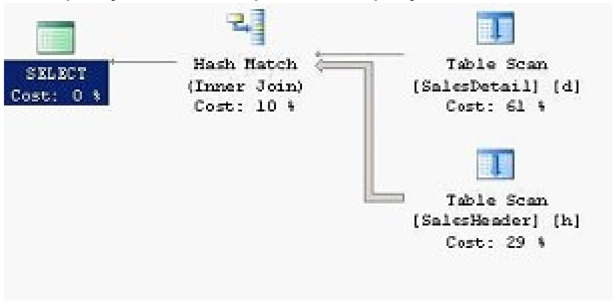
You are a database developer on an instance of SQL Server 2008. Your database contains the **SalesHeader** and **SalesDetail** tables as shown:



The **SalesHeader** and **SalesDetail** tables both contain millions of rows.
You execute the following query against the tables:

```
SELECT ProductID, SpecialOfferIDFROM Detail d INNER JOIN SalesHeader h ON d.
SalesOrderID = h.SalesOrderID WHERE SpecialOfferID = 5;
```

The query execution plan is displayed as follows:



Which action should you take?

A. Create a partitioned view over the table and use it in the query.
B. Create a nonclustered index on the **SpecialOfferID** column of the **SalesDetail** table.
C. Implement page-level compression for the **SalesDetail** table.
D. Rebuild statistics for both tables.

Answer: B
Section: (none)

Explanation/Reference:
In this scenario, a full table scan is performed on the **SalesDetail** table and a table scan is performed on the **SalesDetail** table. Therefore, adding a nonclustered index on the table would likely improve performance for the query.
You should not create a partitioned view over the table and use it in the query. Partitioned views are used when you have similar data stored in multiple tables and want to create a view to allow access to all of the data as if it were stored in a single table. Partitioned views are implemented by creating a view that queries several tables and combines the results using the `UNION ALL` operator. A partitioned view can improve performance and increase availability, but would not be applicable in this scenario. Partitioned views can also be used to query data across multiple servers and provide additional scalability.

You should not implement page-level compression for the **SalesDetail** table. Implementing compression would save storage space for the **SalesDetail** table, but this would not be applicable in this scenario. Page-level compression will not reduce the cost associated with the **SalesHeader** or **SalesDetail** table.
You should not rebuild statistics for both tables. There is no indication that statistics for the table are stale.

QUESTION 82
You are a database developer on an instance of SQL Server 2008. Your database contains the **Task** and **TaskType** tables, which were defined with the following Transact-SQL statements:

```
CREATE TABLE TaskType (TaskTypeID int PRIMARY KEY,Description varchar(40) NOT
NULL);
CREATE TABLE Task (TaskID int IDENTITY(1,1) PRIMARY KEY,Description varchar(50)
NOT NULL,TaskTypeID int NOT NULL FOREIGN KEY REFERENCES TaskType(TaskTypeID),
DateAdded datetime NOT NULL DEFAULT (GETDATE()),DateDue datetime);
```

You need to create a view that makes data from both tables visible. You also want the view to be updateable.
Which action should you take?

A. Create a partitioned view that allows users to update only their respective partitions.
B. Create a view that includes a CTE.
C. Create a view that includes both tables and define `INSTEAD OF` triggers to allow DML operations.
D. Create a parameterized stored procedure and create a view on the returned result set.

Answer: C
Section: (none)

Explanation/Reference:
Usually views are based on a `SELECT` statement and only provide query access to underlying data. However, you can create a view and then use `INSTEAD OF` triggers defined for the view to implement DML functionality. An `INSTEAD OF` trigger fires in place of the operation that fired the trigger. `INSTEAD OF` triggers can be used to prevent the original operation from taking place or to perform additional or alternative actions. In this scenario, you could create a view that made data in both tables available and then create the necessary `INSTEAD OF` triggers that would insert, update, or delete from both tables as needed. When you use this approach, you must write your code to specifically handle any constraints defined on the base tables' columns. Also, for `IDENTITY` columns, such as **TaskID** in the **Task** table, you would use the `@@IDENTITY` function to retrieve the last identity value before inserting or updating a record.
You should not create a partitioned view that allows users to update only their respective partitions. Partitioned views are used when you have similar data stored in multiple tables and want to create a view to allow access to all of the data as if it were stored in a single table. Partitioned views are implemented using the `UNION ALL` operator. For example, if you had three separate tables with an identical structure, you might use the following statement to create a partitioned view that allows users to query data from all three tables:

```
CREATE VIEW PartView AS SELECT * FROM Table1 UNION ALL SELECT * FROM Table2 UNION
ALL SELECT * FROM Table3;
```

You should not create a view that includes a CTE. A Common Table Expression (CTE) would not be useful in this scenario. Using a CTE makes the Transact-SQL code, such as the view's definition in this scenario, more readable than using a subquery. The syntax for creating a CTE is as follows:

```
WITH expression_name [(column_name [,...n])]AS (CTE_query_definition)
```

When defining a CTE, the `WITH` clause specifies the expression name that will be used in the subsequent `SELECT` statement. The `WITH` clause
must contain a column list identifying the available columns, unless all columns in the expression's query have distinct names. After you create the CTE, the statement immediately following the CTE definition can reference the CTE expression by name one or more times as if it were a table or view. Only the columns defined in the CTE expression are accessible. You can also create two CTEs in a single `WITH` clause by separating the expressions with a comma. Within the `WITH` clause, the CTE can also reference itself or another CTE that you previously created. You should also note that only one `WITH` clause is allowed, even if the query defining the CTE contains a subquery. In addition, a CTE query definition cannot contain an `ORDER BY`, `COMPUTE`, `COMPUTE BY`, `INTO`, `FOR XML`, or `FOR BROWSE` clause, or an `OPTION` clause that specifies

query hints.
You should not create a parameterized stored procedure and create a view on the returned result set. Although you might choose to implement parameterized stored procedures to implement DML functionality, you cannot create a view over a result set.

QUESTION 83
Your database contains the **Donor** and **DonorDetail** tables. You are creating a view. The view's definition is a complex query that references the same subquery multiple times. You want to allow users to access to the data, but avoid the code complexity of the view's definition. Which action should you take?

A. Create a partitioned view to implement the subquery.
B. Use a parameterized stored procedure instead of a view.
C. Include a Common Table Expression (CTE) in the view's definition.
D. Create a temporary table containing the subquery results.

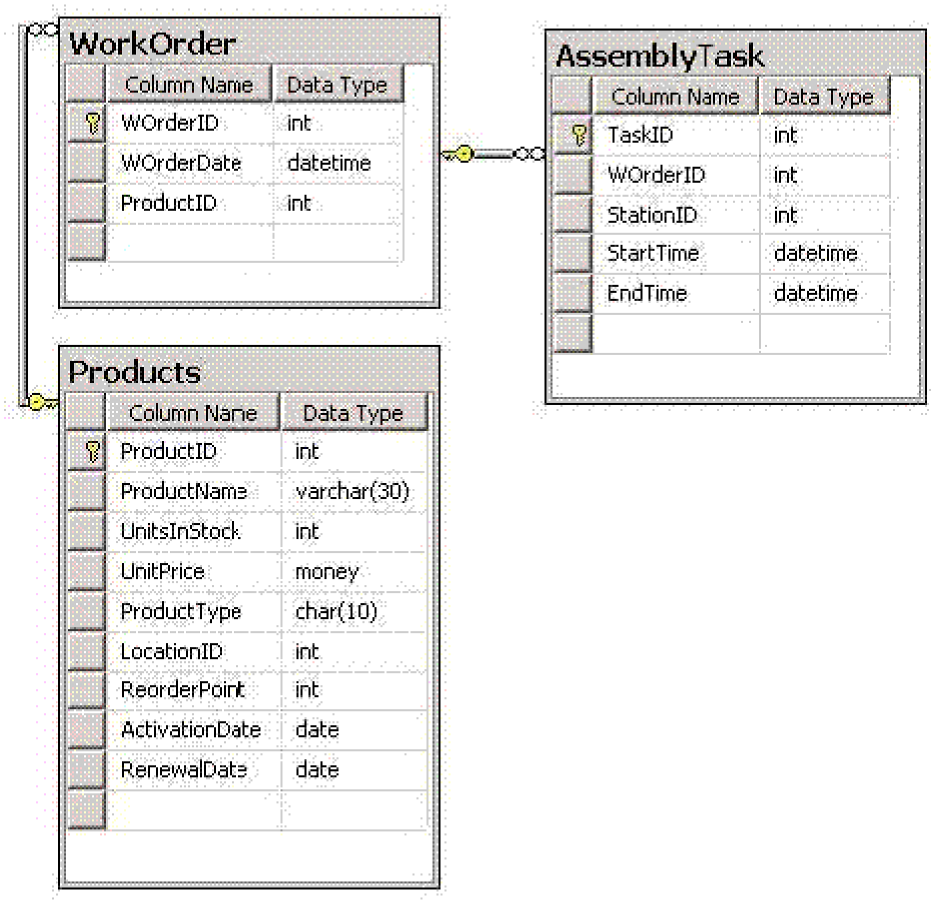
Answer: C
Section: (none)

Explanation/Reference:
You should include a Common Table Expression (CTE) in the view's definition. Using a CTE in the view's definition in this scenario makes the Transact-SQL code more readable than using a subquery. The syntax for creating a CTE is as follows:
```
WITH expression_name [(column_name [,...n])]AS (CTE_query_definition)
```
When defining a CTE, the `WITH` clause specifies the expression name that will be used in the subsequent SELECT statement. A CTE can also be used when defining a view. The `WITH` clause must contain a column list identifying the available columns, unless all columns in the expression's query have distinct names. After you create the CTE, the statement immediately following the CTE definition can reference the CTE expression by name one or more times, as if it were a table or view. Only the columns defined in the CTE expression are accessible.
You can also create two CTEs in a single `WITH` clause by separating the expressions with a comma. Within the `WITH` clause, the CTE can also reference itself or another CTE that you previously created. A CTE definition can only include one `WITH` clause, even if the query defining the CTE contains a subquery. In addition, a CTE query definition cannot contain an `ORDER BY`, `COMPUTE`, `COMPUTE BY`, `INTO`, `FOR XML`, or `FOR BROWSE` clause, or an `OPTION` clause that specifies query hints. You can use CTEs as an alternative to using a view, temporary table, or subquery.
You should not create a partitioned view to implement the subquery. Partitioned views are used when you have similar data stored in multiple tables and want to create a view to allow access to all of the data as if it were stored in a single table. Partitioned views are implemented using the `UNION ALL` operator. For example, if you had three separate tables with identical structures, you might use the following statement to create a partitioned view that allows users to query data from all three tables:
```
CREATE VIEW PartView AS SELECT * FROM Table1 UNION ALL SELECT * FROM Table2 UNION
ALL SELECT * FROM Table3;
```
You should not use a parameterized stored procedure instead of a view. Although using parameterized stored procedures is a recommended practice when you create stored procedures, you do not require a parameterized stored procedure in this scenario, and it would require additional development effort to create one. Parameterized stored procedures are stored procedures that accept input parameter values at runtime. The parameters can then be used within the procedure's code. For example, you might use a parameterized stored procedure and use the input parameters to construct a DML statement rather than accepting a string containing the DML statement. This approach would help to prevent SQL injection attacks.
You should not create a temporary table containing the subquery results. Temporary tables are only available while the table is being used. Local temporary tables are available to a single user until the user disconnects. Global temporary tables are available until all users using the table disconnect. If you attempt to create a view on a temporary table, you will receive an error similar to the following:
```
Msg 4508, Level 16, State 1, Procedure MyView, Line 3Views or functions are not
allowed on temporary tables. Table names that begin with '#' denote temporary
```

```
tables.
```

QUESTION 84
You are a database administrator on an instance of SQL Server 2008. Your **Prod** database contains data relating to your company's manufacturing processes. You are creating an application that will track how products are assembled on the manufacturing floor and the time taken during each assembly task. A portion of your data model is as follows:
(Ckick on Exhibits to view the table structures)
The **AssemblyTask** table must track the time it takes to complete each assembly task. Users will frequently run queries specifying time durations to identify assembly tasks that take minimal time or excessive time to complete. You want to minimize database storage where possible. However, query performance is your primary concern.
Which action should you take?

Exhibit:



WorkOrder

Column Name	Data Type
WOrderID	int
WOrderDate	datetime
ProductID	int

AssemblyTask

Column Name	Data Type
TaskID	int
WOrderID	int
StationID	int
StartTime	datetime
EndTime	datetime

Products

Column Name	Data Type
ProductID	int
ProductName	varchar(30)
UnitsInStock	int
UnitPrice	money
ProductType	char(10)
LocationID	int
ReorderPoint	int
ActivationDate	date
RenewalDate	date

A. Create a persisted computed column to store each task's duration, and create an index on the computed column.
B. Create a dedicated column to store each task's duration and use a DML trigger to populate the column.
C. Create a stored procedure that calculates each task's duration.
D. Create a non-persisted computed column that includes a CLR user-defined function to store each task's duration, and create an index on the
computed column.

Answer: A
Section: (none)

Explanation/Reference:
Computed columns are virtual columns that are not physically stored in the table by default. Each computed column uses the `AS` keyword followed by an expression that evaluates to a value. The expression can contain constants, functions, operators, and references to other columns within the table. The value of the computed column is calculated each time a query referencing it executes. You can also include the optional `PERSISTED` keyword when creating a computed column. When a persisted computed column is created, it is physically stored in the table, and is recalculated each time a column value referenced in the calculation expression is changed. For an index to be created on a computed column, the column must be deterministic and precise. A computed column is considered deterministic if it produces the same value each time it is passed the same values. For example, the `GETDATE()` function returns a different value each time it is called. Therefore, a computed column that referenced the `GETDATE()` function would
be non-deterministic. A computed column is considered precise if it does not perform floating-point calculations using a **float** or **real** data type. For example, a computed column that returns an **int** data type but uses a **float** or **real** data type in its definition is imprecise, and a computed column that returns an **int** data type and does not use a **float** or **real** data type is precise. In this scenario, the calculation for the **Duration** is deterministic and precise. Therefore, you could use either of the following `ALTER TABLE` statements to create the **Duration** column:

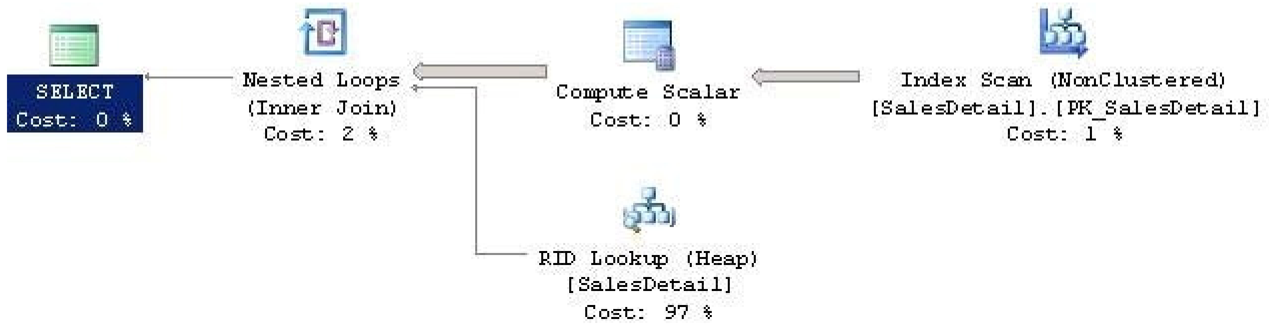
```
ALTER TABLE dbo.AssemblyTaskADD Duration AS EndTime - StartTime PERSISTED;
ALTER TABLE dbo.AssemblyTaskADD Duration AS EndTime - StartTime;
```

Then, you could create an index on the **Duration** column to improve query performance.
You can use the **IsDeterministic** and **IsPrecise** properties with the `COLUMNPROPERTY` function to determine if an expression used in a computed column is deterministic and precise, respectively.
You should not create a dedicated column to store each task's duration and use a DML trigger to populate the column. A DML trigger defines Transact-SQL code that executes when DML statements, such as `INSERT`, `UPDATE`, and `DELETE` statements, are issued on tables or views. Within the body of the trigger, you could include Transact-SQL code to prevent the original operation or perform additional or alternative actions. For the database in this scenario, you could use an `INSTEAD OF` trigger to fire for each insert or update. The trigger could calculate the duration value. A DML trigger might be used if you wanted to log DML errors but continue processing or perform complex validation that roll backs the triggering operation or returns an error message to the caller. If you use triggers to additional actions, you should note that `AFTER` triggers do not fire if the triggering SQL statement fails, for example if constraint checks are not successful. However, in this scenario, performance is a primary concern. Therefore, you should avoid using triggers because they require more overhead.
You should not create a stored procedure that calculates each task's duration. This would not improve query performance because the stored procedure call would have to be included for each query.
You should not create a non-persisted computed column that includes a CLR user-defined function to store each task's duration, and create an index on the computed column. To create an index on a computed column that references a CLR user-defined function, the computed column must be persisted.

QUESTION 85
You are a database developer on an instance of SQL Server 2008. Your **Sales** database contains the following **SalesDetail** table: (Click on Exhibits to view table)
Currently, only a unique nonclustered index exists on the **SalesOrderDetailID** column named **PK_SalesDetail**. You have the following query that is performing poorly:

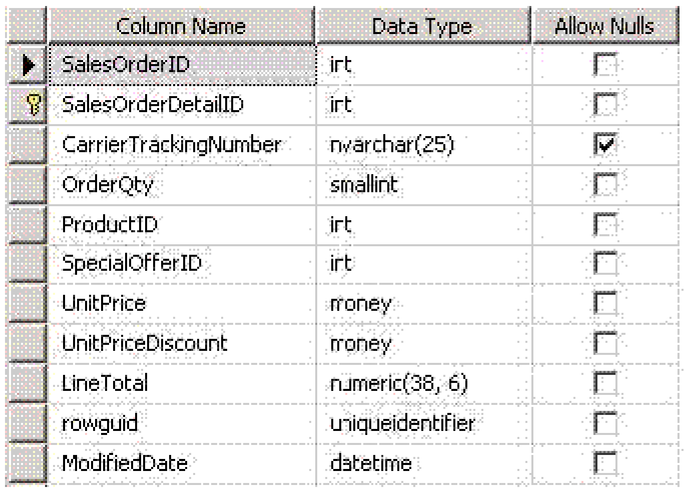
```
SELECT * FROM SalesDetail WITH (INDEX(PK_SalesDetail))WHERE UnitPrice > 10 AND
UnitPrice < 2000 AND CarrierTrackingNumber = '0538-491B-B6';
```

The query generates the following actual execution plan:



What should you do to improve the query's performance?

Exhibit:



Column Name	Data Type	Allow Nulls
SalesOrderID	int	☐
SalesOrderDetailID	int	☐
CarrierTrackingNumber	nvarchar(25)	☑
OrderQty	smallint	☐
ProductID	int	☐
SpecialOfferID	int	☐
UnitPrice	money	☐
UnitPriceDiscount	money	☐
LineTotal	numeric(38, 6)	☐
rowguid	uniqueidentifier	☐
ModifiedDate	datetime	☐

A. Create a plan guide including the `OPTIMIZE FOR` hint.
B. Include the `MAXDOP` query hint in the query.
C. Modify the `WHERE` clause to use the `BETWEEN` operator.
D. Create a nonclustered index on the **CarrierTrackingNumber** and **UnitPrice** columns.

Answer: D
Section: (none)

Explanation/Reference:
In this scenario, the only index that exists is the nonclustered index on the primary key column. The given query uses the **CarrierTrackingNumber** and **UnitPrice** columns in `WHERE` clause conditions. Therefore, you should create a nonclustered index on these columns. This would improve query performance because the optimizer would use an **Index Scan** or an **Index Seek** operation instead of an **RID Lookup**. An **Index Scan** scans the entire nonclustered index, and an **Index Seek** does not. Therefore, to optimize query performance, you might also ensure that an **Index Seek** is performed. In this scenario, you could use the following statement to create a nonclustered index on the two columns:
```
CREATE INDEX IX_TrackingUnitPriceON SalesDetail(CarrierTrackingNumber,
UnitPrice);
```
Then, you could remove the `INDEX` hint in your query so that the optimizer would not be forced to use the **PK_SalesDetail** nonclustered index.
You should not create a plan guide including the `OPTIMIZE FOR` hint. Plan guides can be used to optimize queries without modifying the query directly. Plan guides are helpful when you need to specify query hints but cannot directly access the query, such as when it is embedded within an application. In a plan guide, you include the query to be optimized, and the query hints or a query plan that should be used for optimization. The `OPTIMIZE FOR` query hint forces the optimizer to use a specific parameter value when optimizing the

query.
You should not include the `MAXDOP` query hint in the query. The `MAXDOP` query hint specifies an integer value that identifies the number of processors on which a single query may run. If the query is run on a computer with a single processor, the query hint is ignored. You can also accomplish this by setting the **max degree of parallelism** configuration option on the SQL Server instance, but this setting would affect all queries.
You should not modify the `WHERE` clause to use the `BETWEEN` operator. The query would still access the same underlying data and would use a lookup operator to do so.

QUESTION 86
You are a database developer on an instance of SQL Server 2008. Your **Prod** database contains all your manufacturing-related data, and is accessed by many different applications.
The **Transactions** table is populated by a third-party application, and many internal systems and external systems then utilize the captured data. The **Transactions** table has multiple columns, including an **xml** data type column named **TransItems**.
The **Transactions** table has a primary key defined on the **TransID** column, and has only two other indexes defined on it as follows:
A nonclustered index on the **TransStatus** column
A nonclustered, composite index on the **TransStation** and **TransTask** columns

You have a stored procedure named **GetItems** that is frequently used. The stored procedure accepts several input parameters and queries the **TransItems** column in the **Transactions** table based on the specified input. The query utilizes XML methods to extract XML fragments that meet the specified criteria. You want to ensure the queries execute as efficiently as possible.
What should you do?

A. Create a primary XML index on the **TransItems** column.
B. Add code to the stored procedure to shred the data found in the **TransItems** column into columns in another table. Create a nonclustered, composite index on the new table.
C. Create a VALUE secondary index on the **TransItems** column.
D. Create a PATH secondary index on the **TransItems** column.

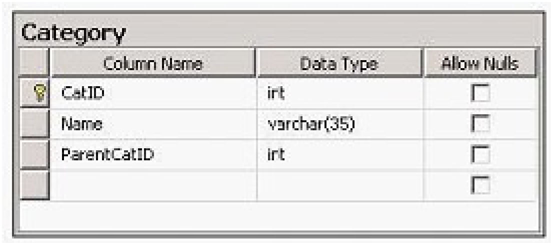
Answer: A
Section: (none)

Explanation/Reference:
. XML indexes can be used to optimize queries on an **xml** column. The optimizer uses these indexes differently than regular indexes, but they can often improve performance based on the given workload. XML indexes can be either primary or secondary. A primary index must be created first. Each **xml** column in a table can have only one primary XML index defined. Primary indexes can be created on **xml** columns containing either typed or untyped XML. The primary XML index uses all paths and values in the **xml** column to create an index with a B-Tree structure. You create a primary XML index using the `CREATE PRIMARY XML INDEX` statement. In this scenario, you could create a primary XML index on the **TransItems** column using the following statement:
```
CREATE PRIMARY XML INDEX PXML_TransItems ON Transactions(TransItems);
```
You can also use secondary XML indexes to improve performance of specific queries. A secondary XML index can be created on an **xml** column only after a primary XML index has been created. When creating a secondary index, you omit the `PRIMARY` keyword and specify the primary XML index in the `USING XML INDEX` clause. Secondary XML indexes can be one of three types: PATH, VALUE, or PROPERTY. The type of secondary index is specified by the `FOR` clause. VALUE secondary XML indexes can improve performance if queries are often performed searching for specific values without specified paths. PATH secondary XML indexes may improve performance when path expressions are often used in queries. PROPERTY secondary XML indexes can improve performance for queries that use the **value()** method to retrieve one or more values. For example, in this scenario, you might create a secondary XML index using the following statement:

```
CREATE XML INDEX SXML_TransItems ON Transactions(TransItems)USING XML INDEX
PXML_TransItems FOR VALUE;
```

You should not add code to the stored procedure to shred the data found in the **TransItems** column into columns in another table and create a composite index on the new table. In this scenario, the **TransItems** column is used by other applications, so it is best to leave it in XML format rather than shredding it into relational data. In addition, this solution would likely decrease performance of the **GetItems** stored procedure.
You should not create a VALUE secondary index or a PATH secondary index on the **TransItems** column because to create a secondary XML index, a primary XML index must first be created. You must specify the previously created primary XML index in the `USING XML INDEX` clause. If you attempt to create a secondary index omitting the `USING XML INDEX` clause, the statement will generate a syntax error.

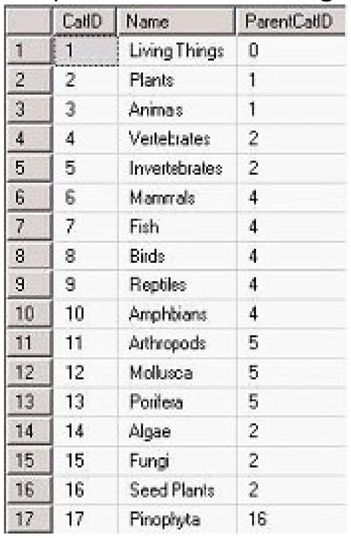
QUESTION 87
You are a database developer on an instance of SQL Server 2008. You maintain a **Research** database that contains the following **Category** table:



Category

Column Name	Data Type	Allow Nulls
CatID	int	☐
Name	varchar(35)	☐
ParentCatID	int	☐
		☐

The **Category** table contains a hierarchy of living things to the sub-species level. Each row in the **Category** table has a **ParentCatID** value that corresponds to the item's parent classification in the hierarchy. The parent-child relationship represented by the **CatID** and **ParentCatID** columns is nested more than 20 levels. A sample of data in the **Category** is shown as follows:



	CatID	Name	ParentCatID
1	1	Living Things	0
2	2	Plants	1
3	3	Animas	1
4	4	Vertebrates	2
5	5	Invertebrates	2
6	6	Mammals	4
7	7	Fish	4
8	8	Birds	4
9	9	Reptiles	4
10	10	Amphibians	4
11	11	Arthropods	5
12	12	Mollusca	5
13	13	Porifera	5
14	14	Algae	2
15	15	Fungi	2
16	16	Seed Plants	2
17	17	Pinophyta	16

You want to query the **Category** table and return a list of all living things, but only down to five levels of nesting. You want to accomplish this with a single query. You want to minimize the complexity of the query, but provide optimal performance.
Which action should you take?

A. Create a query that contains four subqueries.
B. Create a recursive CTE.
C. Create a query that uses the `CROSS APPLY` clause.
D. Create a query that returns a calculated **hierarchyid** value.

Answer: B
Section: (none)

Explanation/Reference:
A Common Table Expression (CTE) can be used to make Transact-SQL code more readable when a query needs to reference the same result set multiple times. You can define a CTE by including the `WITH` clause with a query. When defining a CTE, the `WITH` clause specifies the expression name that will be used in the statement that immediately follows the CTE definition. The statement immediately following the CTE definition can use the expression one or more times, as if it were a table or view. The `WITH` clause must contain a column list identifying the available columns, unless all columns in the expression's query have distinct names. The syntax for creating a CTE is as follows:
```
WITH expression_name [(column_name [,...n])]AS (CTE_query_definition)
```
After you create the CTE, the statement immediately following the CTE definition can reference the expression, but only the columns defined in the CTE expression are accessible. CTEs can be used as an alternative to a view, temporary table, or subquery. The query for a CTE definition can reference the original CTE. In this scenario, you could use the following query:
```
WITH CatCTE AS (SELECT CatID, Name, ParentCatIDFROM CategoryWHERE ParentCatID = 0
UNION ALL SELECT p.CatID, p.Name, p.ParentCatIDFROM Category pINNER JOIN CatCTE c
ON c.CatID = p.ParentCatID)SELECT * FROM CatCTE OPTION (MAXRECURSION 4);
```
This CTE definition references itself on the right side of the `INNER JOIN`. This causes the CTE to be recursive. The first `SELECT` statement in the CTE definition returns the row at the top of the hierarchy, which has a **ParentCatID** value of 0. This CTE query is referred to as an anchor member because it does not reference the CTE definition. The `SELECT` statement following the `UNION ALL` operator is referred to as a recursive member because it references the CTE definition. When creating a recursive CTE, forward referencing is not allowed. This means that the anchor member must be specified first. This statement uses the `UNION ALL` operator to combine the results of the first and second `SELECT` statements. You can also use the `UNION`, `INTERSECT`, or `EXCEPT` operators, but the `UNION ALL` operator must always be specified after the last anchor member before recursive members. Because set operators are used, both `SELECT` statements must contain the same number of columns, and columns must have the same data type. In this scenario, the `SELECT` statement following the `UNION ALL` operator joins the original table with the result of the CTE. This joins each item with its respective parent, continuing top-down through the hierarchy.
The statement that follows the CTE expression definition specifies the `MAXRECURSION` hint to limit recursion to four levels. This means that the recursive query can only call itself four times. This traverses down five levels, one for the top-level item and one for each recursion. In this scenario, when the recursion level maximum is exceeded, an error is generated, but the result set up to that point is returned.
You should not create a query that contains four subqueries. This would add complexity to the query, and would not optimize performance.
You should not create a query that uses the `CROSS APPLY` clause. The `APPLY` clause is used in the `FROM` clause of a query to join a table to a table-valued function. The table-valued function is called for each row returned by the outer query. The `APPLY` clause allows you to easily call a table-valued function for each row returned by a query. The `OUTER APPLY` clause returns all rows from the outer query, even if the row does not return a value for the table-valued function. The `CROSS APPLY` clause returns only the outer query rows for which the table-valued function returns a value.
You should not create a query that returns a calculated **hierarchyid** value. The **hierarchyid** data type is a special variable-length, CLR-supplied data type that can be used to represent hierarchical data. Defining a **hierarchyid** data type column allows you to use various system-supplied methods, such as **GetRoot**, **GetDescendant**, **GetLevel**, **Read**, and **Write**, which allow you to perform tasks on the hierarchy. Each value stored in the table is an internal binary value that represents the row's position in the hierarchy. In this scenario, you could represent the relationship by implementing a **hierarchyid** column in the table, but you would have to modify the table structure.

QUESTION 88
You are a database developer on an instance of SQL Server 2008. You maintain a table, named **EventDetails**, containing information on scheduled events defined as follows:
```
CREATE TABLE EventDetails (EventID int IDENTITY(1,1),EventName varchar(50),
Description varchar(400),FacilitatorID int,EventDt datetime,CONSTRAINT PK_EventID
PRIMARY KEY CLUSTERED (EventID));
```
You execute the following Transact-SQL statement:
```
CREATE FULLTEXT CATALOG EventFTCatalogAS DEFAULT;
CREATE FULLTEXT INDEX ON dbo.EventDetails (Description)KEY INDEX PK_EventID ON
EventFTCatalogWITH CHANGE_TRACKING AUTO;
```
You want to display the names of all events in the **EventDetails** table that have a description containing either the word introductory or a word with the same meaning as introductory.
Which three actions should you take? (Choose three. Each correct answer represents part of the solution.)

A. Create a query that uses the `LIKE` operator.
B. Modify the thesaurus file to include all words with the same meaning as introductory.
C. Rebuild the full-text index on the **EventDetails** table.
D. Enable full-text support for the database.
E. Create a query that uses the `CONTAINS` predicate with the `FORMS OF THESAURUS` clause.
F. Reload the thesaurus file using the **sys.sp_fulltext_load_thesaurus_file** system stored procedure.
G. Create a query that uses the `CONTAINSTABLE` function with the `ISABOUT` clause.

Answer: BEF
Section: (none)

Explanation/Reference:
To accomplish the desired objectives, you should perform the following tasks:
Modify the thesaurus file to include all words with the same meaning as introductory.
Reload the thesaurus file using the **sys.sp_fulltext_load_thesaurus_file** system stored procedure.
Create a query that uses the `CONTAINS` predicate with the `FORMS OF THESAURUS` clause.

SQL Server 2008 has default thesaurus files that can be used with full-text searches to search for synonyms. These thesaurus files are XML files that can be modified to contain the desired synonyms or pattern replacements. Thesaurus files are used with queries that use the `FREETEXT` predicate and the `FREETEXTTABLE` function, and with queries that use the `CONTAINS` predicate and the `CONTAINSTABLE` function with the `FORMS OF THESAURUS` clause. A global thesaurus file, named **tsGlobal.xml**, and a thesaurus file for each language are located in the **SQL_Server_install_path\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\FTDATA\** folder. These default thesaurus files have XML elements commented out initially, but you can modify them as required.
In this scenario, you wanted to display the names of all events in the **EventDetails** table with a description containing the word introductory or a word with the same meaning as introductory. To accomplish this, you should first modify the thesaurus file to include the desired synonyms for the word introductory. For example, you would modify the thesaurus file as follows to identify the word beginning as a synonym for the word introductory:
```
<XML ID="Microsoft Search Thesaurus"> <thesaurus xmlns="x-schema:tsSchema.xml">
     <diacritics_sensitive>0</diacritics_sensitive><expansion><sub>Internet
Explorer</sub><sub>IE</sub><sub>IE5</sub></expansion><replacement><pat>NT5</
pat><pat>W2K</pat><sub>Windows 2000</sub></
replacement><expansion><sub>introductory</sub><sub>beginning</sub></expansion></
thesaurus></XML>
```
After modifying the thesaurus file, you should reload it using the **sys.sp_fulltext_load_thesaurus_file** system stored procedure for the changes to take effect. For example, the following statement would reload the English thesaurus file:
```
EXEC sys.sp_fulltext_load_thesaurus_file 1033;
```
Finally, you should create a query that uses the modified thesaurus file. The following query uses the `CONTAINS` predicate with the `FORMS OF THESAURUS` clause to return the desired results:
```
SELECT EventName FROM dbo.EventDetails WHERE CONTAINS (Description, 'FORMSOF
(THESAURUS, introductory)');
```
You should not create a query that uses the `LIKE` operator. The `LIKE` operator is used to identify values that contain a specific character pattern, not synonyms.

You should not rebuild the full-text index on the **EventDetails** table. In this scenario, you specified the `WITH CHANGE_TRACKING AUTO` clause with the `CREATE FULLTEXT INDEX` statement. This configures automatic population of the full-text index. Each time the **Description** column of the **EventDetails** table is updated, the changes are also made to the full-text index. Therefore, there is no need to rebuild the full-text index.
You should not enable full-text support for the database. In previous versions of SQL Server, full-text support had to be enabled, but with SQL Server 2008, full-text search support is enabled for all user databases by default.
You should not create a query that uses the `CONTAINSTABLE` function with the `ISABOUT` clause. The `ISABOUT` clause is used to search for words with different weightings, not for word synonyms. For example, the following query uses the `CONTAINSTABLE` function in the `FROM` clause of a query to search the **EventDetails** table for the words novice, beginning, and introductory with a different weighting assigned to each word, and returns the rows with the highest rank first:

```
SELECT * FROM EventDetails AS e INNER JOIN CONTAINSTABLE(EventDetails,
Description, 'ISABOUT (novice weight(.8), beginning weight(.4),introductory
weight (.1))') AS kON e.EventID = k.[KEY]ORDER BY k.[RANK] DESC;
```

QUESTION 89
You are a database developer on an instance of SQL Server 2008. Your **Prod** database contains an **InventoryHistory** table that contains a historical record of all inventory transactions that have occurred in your warehouse.
You need to allow users to access specific data from the **InventoryHistory** table, but not to make updates. You want to allow users to provide three different search criteria values and retrieve historical inventory information based on the values provided. Data will be retrieved from the **InventoryHistory** table and other lookup tables based on the user's specified criteria. Users must then be able to use this data to perform additional analysis.
You want to accomplish this with minimal development effort.
Which action should you take?

A. Create a recursive scalar user-defined function.
B. Create an inline table-valued function.
C. Create a partitioned view on the **InventoryHistory** table.
D. Create a multi-statement table-valued function.

Answer: B
Section: (none)

Explanation/Reference:
. In SQL Server 2008, user-defined functions may be one of three types: scalar-valued, inline table-valued, or multi-statement table-valued. Both inline table-valued and multi-statement table-valued functions can be used to return a result set as a **table** data type. Inline table-valued functions can execute only one SQL statement. Multi-statement table-valued functions can execute multiple SQL statements as well as include other procedural code. An inline table-valued function can be an alternative to creating a view, and provides the benefit of being able to specify parameters within the `SELECT` statement. Inline views are sometimes referred to as parameterized views. In this scenario, because the user-defined function will not include multiple SQL statements and only needs to return rows based on the parameters it was passed, you should use an inline table-valued function.
To create an inline table-valued function, use the `CREATE FUNCTION` statement and specify a **table** data type as the return value. Unlike a multi-statement table-valued function, an inline table-valued function does not have to define the structure of the table being returned. The structure of the returned table is derived by the `SELECT` statement included within the function. For example, the following statement would create an inline table-valued function that accepts two parameters and returns a table of data based on a query of two tables:

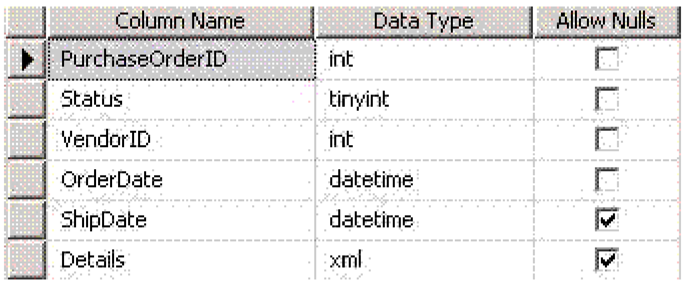
```
CREATE FUNCTION dbo.GetData (@typedesc varchar(10), @loc int)RETURNS table AS
RETURN (SELECT i.ProductName, t.Description FROM InventoryHistory AS iJOIN
PTLookup AS t ON t.TypeID = i.TypeIDWHERE t.Description = @typedesc ANDi.LocID =
@loc);
```

This would create an inline table-valued function named **dbo.GetData** that users could use to query the **InventoryHistory** table and the **PTLookup** lookup table. Users would be able to call the function from within a `FROM` clause, passing it the search parameters to return the data and perform additional analysis.
You should not create a recursive scalar user-defined function. A scalar user-defined function returns a single scalar value, and in this scenario, you want users to access a result set.
You should not create a multi-statement table-valued function because this would require more development effort. With a multi-statement table-valued function, you must explicitly define the table being returned, but with an inline table-valued function, you do not.
You should not create a partitioned view on the **InventoryHistory** table. A partitioned view is used when you have similar data stored in multiple tables and want to create a view to allow access to all of the data as if it were stored in a single table. Partitioned views are implemented using the `UNION ALL` operator. For example, if you had three separate tables with an identical structure, you might use the following statement to create a partitioned view that allows users to query data from all three tables:

```
CREATE VIEW PartView AS SELECT * FROM Table1 UNION ALL SELECT * FROM Table2 UNION
ALL SELECT * FROM Table3;
```

QUESTION 90
You are a database developer on an instance of SQL Server. Your **POMaster** table is defined as follows:
The **Details** column contains XML in the format shown in the exhibit. (Click the **Exhibit(s)** button.)
You want to insert rows into another table, named **PODetails**, that contain each purchase order line for selected purchase orders. You want to
accomplish this with minimal effort and memory consumption and allow for code reuse.

Exhibit:



Column Name	Data Type	Allow Nulls
PurchaseOrderID	int	☐
Status	tinyint	☐
VendorID	int	☐
OrderDate	datetime	☐
ShipDate	datetime	☑
Details	xml	☑

A. Create a stored procedure that accepts an **xml** input parameter and uses `OPENXML`.
B. Create a user-defined function that accepts an **xml** input parameter and returns a **table** data type.
C. Use a `SELECT` statement that uses the **query()** XML method in the `SELECT` list.
D. Create a stored procedure that accepts an **xml** input parameter and uses XML methods.

Answer: D
Section: (none)

Explanation/Reference:
You should create a stored procedure that accepts an **xml** input parameter and uses XML methods. The stored procedure would select the desired
elements and attributes using the **value()** method in the `SELECT` list with the **nodes()** method specified in the `FROM` clause. For example, in this
scenario you might use the following stored procedure to insert rows into the **PODetails** table:

```
CREATE PROCEDURE usp_InsertPOLines (@xml xml)
AS
BEGIN
INSERT INTO PODetails (InvID, Quantity, Cost, TaxFlag)
SELECT t.c.value('@InvID', 'int'),
t.c.value('@Quantity', 'int'),
t.c.value('@Cost', 'money'),
t.c.value('@Taxable', 'bit') AS TaxFlag
FROM @xml.nodes('/POLines/POLine') AS t(c)
END
```

The **nodes()** method accepts an XQuery string and returns the specified nodes as a result set. In this scenario, the **nodes()** method returns a result
set that contains one row for each purchase order line. Then, in the `SELECT` list, the table alias defined is used with the **value()** method to return the
details for each line. The **value()** method accepts two arguments. The first argument is a string XQuery expression, and the second argument is a
string containing a SQL Server data type. The **value()** method extracts a value using the XPath expression and returns it as the specified type. You
could use the **value()** method to extract the desired data as a specified data type to be inserted into the **PODetails** table.
You should not use a `SELECT` statement that uses the **query()** XML method in the `SELECT` list. The **query()** method is used to query and retrieve
XML elements and attributes from an XML instance as untyped XML. The method accepts a string XQuery expression that determines which
elements and element attributes are extracted. In this scenario, you needed to return the data from the XML with specific data types, not in XML
format.
You should not create a user-defined function that accepts an `xml` input parameter and returns a `table` data type. You would still need to insert the
value returned by the function into a table.
You should not create a stored procedure that accepts an **xml** input parameter and uses `OPENXML`. Although you can use `OPENXML` to shred XML
into relational data, `OPENXML` is more memory-intensive, and in this scenario, you wanted to minimize memory consumption. `OPENXML` is a rowset
provider function that creates a relational view of data contained in an XML document. `OPENXML` can be used in place of a table or view in a
`SELECT` statement. To use `OPENXML`, you first call the **sp_xml_preparedocument** system stored procedure to return a document handle. Then,
you include a query that contains the `OPENXML` function in the `FROM` clause and include a `WITH` clause to identify the columns in the output. After
the data has been extracted from the XML document, you call the **sp_xml_removedocument** stored procedure to remove the internal document
handle and free resources.

QUESTION 91
You are a database developer on an instance of SQL Server 2008. Your **Sales** database contains sales-related data for your company.
You need to create a stored procedure that will be used by several in-house applications. Your development team will implement the stored procedure using **DataReader** objects. The stored procedure must query multiple tables in the **Sales** database using a complex join, and return the query results to the caller.
What should you do?

A. Create a stored procedure that accepts scalar input values and returns a result set.
B. Create a CLR stored procedure that accepts multiple scalar input parameters.
C. Create a stored procedure that accepts multiple scalar input parameters and returns data using an `OUTPUT` parameter.
D. Create a stored procedure that accepts a table-valued input parameter and returns a table-valued output parameter.

Answer: A

Section: (none)

Explanation/Reference:
The result set that is returned can then be accessed by the in-house applications. A stored procedure can return an entire result set by including a `SELECT` statement within the stored procedure. For example, you might use the following `CREATE PROCEDURE` statement to return a single result set to the caller:

```
CREATE PROCEDURE usp_GetDonorList (@DonorID int)AS SELECT DonorID,FirstName,
LastName,Amount FROM Donors WHERE DonorID = @DonorIDORDER BY Amount DESC;
```

You should not create a CLR stored procedure that accepts multiple scalar input parameters. There is no need to use a CLR stored procedure in this scenario. A CLR stored procedure is a stored procedure created using a .NET Framework language and then registered with the database so it can be called from Transact-SQL. CLR stored procedures should be used to implement complex logic or functionality that is not inherently supported in the database, such as logic that requires the use of language constructs available in a .NET Framework language. Although a CLR stored procedure could work in this scenario, the option of creating a CLR stored procedure that accepts multiple scalar input parameters is incorrect because this option does not return a result set.
You should not create a stored procedure that accepts multiple scalar input parameters and returns data using an `OUTPUT` parameter. You would use an `OUTPUT` parameter if you wanted to return only a few values rather than the entire query result. For example, you might use the following statement to create a stored procedure named **usp_MyProc** that that returns a single output parameter containing the total number of rows in the **MyTable** table for a specified **ID** value:

QUESTION 92
You are a database developer on an instance of SQL Server 2008. In your **Prod** database, you have purchasing-related tables that contain product, vendor, and purchasing data. You want extract data from several of these tables and display it in XML format. You want to accomplish this with the minimum development effort. Which action should you take?

A. Create a stored procedure that queries the tables and returns an**xml** data type.
B. Use the `FOR XML` clause in a query that joins the required tables.
C. Create a Transact-SQL script that uses the **sp_xml_preparedocument** system stored procedure and then inserts the XML into an **xml** data type column.
D. Implement a query that uses the `OPENXML` function in the `FROM` clause

Answer: B
Section: (none)

Explanation/Reference:
You should use the `FOR XML` clause in a query that joins the required tables. The `FOR XML` clause specifies that the result of the `SELECT` statement should be returned in XML format. You can specify one of the following modes with the `FOR XML` clause:
`RAW`: A single `<row>` element will be generated for each row in the rowset. Each non-null column value generates an attribute with the name identical to the column's name or the column's alias.
`AUTO`: Nested elements are returned using the columns specified in the `SELECT` list. Each non-null column value generates an attribute named according to the column name or column alias. The element nesting is based on the order in which the columns are specified in the `SELECT` list.
`EXPLICIT`: Each row in the rowset can be defined in detail, including attributes and elements.
`PATH`: Each row in the rowset can be defined in detail, but column names and column aliases are specified as XPath expressions.

You can also include the `ELEMENTS` option with the `FOR XML` clause. This will return columns in the `SELECT` list as subelements, rather than as attributes. Each table specified in the `FROM` clause will be represented by a separate element, and each column from that table will appear as a subelement of that element. Tables specified first will constitute higher-level elements of the hierarchy, and if specified, column aliases will be used as element names. For example, in this scenario, you might use the following query to extract data from the **Product**, **ProductVendor**, and **Vendor** tables in XML format:

```
SELECT p.Name As Product, v.Name AS Vendor, p.ProductIDFROM Production.Product
pJOIN Purchasing.ProductVendor pvON p.ProductID = pv.ProductIDJOIN Purchasing.
Vendor vON pv.VendorID = v.VendorIDFOR XML RAW, ELEMENTS;
```

The query would display the XML in the following format:

```
<row> <Product>Product A</Product><Vendor>VirtuArt, Inc.</Vendor><ProductID>1</
ProductID></row><row> <Product>Product B</Product><Vendor>NuTex Corporation</
Vendor><ProductID>879</ProductID></row><row> <Product>Product C</
Product><Vendor>VirtuArt, Inc.</Vendor><ProductID>712</ProductID></row><row>
<Product>Product D</Product><Vendor>InterConn</Vendor><ProductID>2</ProductID></
row>
```

You should not implement a query that uses the `OPENXML` function in the `FROM` clause, or create a Transact-SQL script that uses the **sp_xml_preparedocument** system stored procedure and then inserts the XML into an **xml** data type column. `OPENXML` is a rowset provider function that creates a relational view of data contained in an XML document. This function can be used in `SELECT` statements where a table or view would be specified to extract data from an XML document. One of the function's required parameters is an integer used as a handle to the internal representation of the XML document. Therefore, before you can extract data from an XML document, you must call the **sp_xml_preparedocument** stored procedure to create and return the document handle. After the data has been extracted from the XML document by a `SELECT` query with the `OPENXML` function, you would call the **sp_xml_removedocument** stored procedure to remove the internal representation of the document and free resources.
You should not create a stored procedure that queries the tables and returns an**xml** data type. In this scenario, you wanted to display the data in XML format. Therefore, you would still have to display the result of the returned **xml** data type.

QUESTION 93
You maintain a large production database on an instance of SQL Server 2008. The database is used by a several third-party applications. One of the applications includes a stored procedure defined as follows:

```
CREATE PROCEDURE Sales.GetSalesByTerritory (@Territory nvarchar(30))AS BEGIN
SELECT * FROM Sales.SalesOrderHeader AS h, Sales.Customer AS c,Sales.
SalesTerritory AS tWHERE h.CustomerID = c.CustomerID AND c.TerritoryID = t.
TerritoryIDAND Name = @TerritoryEND;
```

Most sales orders are taken in the Northeast territory, and the stored procedure is performing poorly because it was originally optimized for the Southwest territory.
You want to optimize performance of the stored procedure without modifying any third-party code.
Which action should you take?

A. Create a template plan guide that includes the `RECOMPILE` hint.
B. Create a nonclustered index on the **Name** column of the **SalesTerritory** table.
C. Create an object plan guide that includes the `OPTIMIZE FOR` hint.
D. Partition the **SalesOrderHeader** table on the **TerritoryID** column.

Answer: C
Section: (none)

Explanation/Reference:
In this scenario, the stored procedure was optimized for a territory **Name** value of Southwest. However, most sales orders are taken in the Northeast territory. Therefore, performance would be improved if the query were optimized using a value of Northeast for the territory name. You can use the `OPTIMIZE FOR` query hint to optimize a query for specific parameter values. However, in this scenario, the stored procedure resides in a third-party application that you do not want to modify. Therefore, you should create a plan guide. Plan guides can be used to optimize queries without modifying the statement directly. In a plan guide, you include the query to be optimized, and either an `OPTION` clause with query hints or a query plan that should be used during optimization. You create plan guides using the **sp_create_plan_guide** system stored procedure, which

accepts the following parameters:
`@name` - Specifies a unique plan guide name.
`@stmt` - Specifies the Transact-SQL statement or batch associated with the plan guide.
`@type` - Specifies the type of plan guide. Valid values are `OBJECT`, `SQL`, and `TEMPLATE`.
`@module_or_batch` - Specifies the module name if the plan guide applies to a specific module, such as a stored procedure.
`@params` - Specifies an optional list of parameters to be used for SQL and template plan guide types.
`@hints` -Specifies an `OPTION` clause that includes hints to be used during optimization.

After you create a plan guide for a SQL statement, SQL Server matches the Transact-SQL statement to the plan guide when the query executes and uses the specified `OPTION` clause hints or query plan for the statement. In this scenario, you wanted to optimize the query in the stored procedure for a specific parameter value. You could use the following statement to create the plan guide:

```
sp_create_plan_guide@name = N'SalesByTerritoryGuide',@stmt = N'SELECT * FROM
Sales.SalesOrderHeader AS h,Sales.Customer AS c,Sales.SalesTerritory AS tWHERE h.
CustomerID = c.CustomerID AND c.TerritoryID = t.TerritoryIDAND Name =
@Territory',@type = N'OBJECT',@module_or_batch = N'Sales.GetSalesByTerritory',
@params = NULL,@hints = N'OPTION (OPTIMIZE FOR (@Territory = N''Northeast''))'
```

When the query executes, it will be optimized using an `@Territory` parameter value of 'Northeast'. To view details of plan guides created in a database, you can query **sys.plan_guides**.
You should not create a template plan guide that includes the `RECOMPILE` hint. The `RECOMPILE` hint is used to force recompilation, which was not applicable in this scenario. In addition, template guides are used to override the `PARAMETERIZATION` database setting for a group of queries, not to optimize a single query for a specific parameter value.
You should not create a nonclustered index on the **TerritoryName** column of the **SalesTerritory** table. In this scenario, you needed to optimize the
query for a specific parameter value. Creating a nonclustered index would not provide this.
You should not partition the **SalesOrderHeader** table on the **TerritoryID** column. Partitioning makes large tables easier to manage. Partitioning also speeds up access because you can access smaller subsets of table's data more quickly and maintain the data integrity, as well as spread the partitions across file groups to improve I/O performance.

QUESTION 94
You have to validate an email address using regular expressions and some sophisticated logic, What is the best solution?

A. Create a user defined function in CLR assembly
B. Create a stored procedure
C. Create a trigger
D. use email datatype

Answer: A
Section: (none)

Explanation/Reference:
http://msdn.microsoft.com/en-us/library/ms186755.aspx

http://www.mssqltips.com/tip.asp?tip=1672

http://msdn.microsoft.com/en-us/magazine/cc163473.aspx

QUESTION 95

You are a database developer on an instance of SQL Server 2008. You have a table named **DonorHistory** that contains a historical record of donations that contains over a million rows. You want to use some rows in the **DonorHistory** table to populate a special table of large donations. Your **LargeDonations** table needs store additional information calculated based on the columns in the **DonorHistory** table and other demographic data stored for each donor in a third table, the **DonorDemographic** table. You want to load the data into the **DonorHistory** table as quickly as possible and minimize network traffic during the load. Which actions should you perform? (Choose all that apply. Each correct answer represents a portion of the solution.)

A. Create a user-defined **table** data type.

B. Create a Common Table Expression that includes a `SELECT...INTO` and pass the CTE as an input parameter to a stored procedure.

C. Create a stored procedure that accepts multiple scalar-valued input parameters and performs a single `INSERT`.

D. Create a temporary table containing the desired data and a recursive stored procedure that accepts an array as a parameter and performs a
single `INSERT`.

E. Create a stored procedure that accepts a table-valued input parameter that performs a single `INSERT`.

Answer: AE
Section: (none)

Explanation/Reference:
SQL Server 2008 allows you to use pass table-valued input parameters to stored procedures or user-defined functions. Using table-valued parameters allows you to pass multiple rows of data to a stored procedure or function. This will minimize the number of round-trips to the server. To create a stored procedure that accepts a table-valued parameter as an input parameter, you first define a user-defined data type (UDT) using the `CREATE TYPE...AS TABLE` statement. Then, you can declare a variable of that type, initialize the variable by filling the table with values, and pass the variable as an input parameter to a stored procedure. The stored procedure can then use the **table** variable. For example, you could use the following Transact-SQL statement to define a user-defined **table** type named **MyTableType**:

```
CREATE TYPE MyTableType AS TABLE ( DonorID int,Date datetime,Type char(10),Amount
money);
```

After creating the UDT, you could create a procedure that accepts a parameter of that type using the following Transact-SQL:

```
CREATE PROCEDURE dbo.usp_BuildLargeDonations (@tableparm dbo.MyTableType
READONLY)AS BEGIN
-- Code to perform necessary actions to calculate values to insert
-- into LargeDonations using the table-valued input parameter
SELECT * FROM @tableparmEND GO
```

Table-valued parameters in a stored procedure can only be input parameters, and cannot be modified within the stored procedure. Therefore, they must include the `READONLY` clause in the parameter list. The given `CREATE PROCEDURE` statement creates a stored procedure named **usp_BuildLargeDonations**, which accepts a table-valued parameter. After creating the stored procedure, you could declare and initialize a **table** variable to pass as the input parameter using a `DECLARE` statement, initialize the variable, and pass the variable to the **usp_BuildLargeDonations** stored procedure. For example, you could use the following Transact-SQL to declare and initialize a **table** variable and execute the **usp_BuildLargeDonations** stored procedure:

```
DECLARE @parm AS MyTableType;
INSERT INTO @parm (DonorID, Date, Type, Amount)SELECT DonorID, Date, Type, Amount
FROM DonorHistoryWHERE Amount > 10000;
EXEC usp_BuildLargeDonations @parm;
```

You should not create a Common Table Expression (CTE) that includes a `SELECT...INTO` and pass the CTE as an input parameter to a stored procedure because a CTE cannot contain an `INTO` clause and cannot be passed as a parameter to a stored procedure. A CTE can be used to make Transact-SQL code more readable when a query needs to reference the same result set multiple times. You can define a CTE by including the `WITH` clause with a query. The `WITH` clause specifies the expression name that will be used in the statement that immediately follows the CTE definition. The statement immediately following the CTE definition can use the expression one or more times, as if it were a table or view. The `WITH` clause must contain a column list identifying the available columns, unless all columns in the expression's query have distinct names. The syntax for creating a CTE is as follows:

```
WITH expression_name [(column_name [,...n])]AS (CTE_query_definition)
```

After you create the CTE, the statement immediately following the CTE definition can reference the expression, but only the columns defined in the CTE expression are accessible. CTEs can be used as an alternative to using a view, temporary table, or subquery.
You should not create a stored procedure that accepts multiple scalar-valued input parameters and performs a single `INSERT`. This would not offer the best performance because you would have to call the stored procedure multiple times to insert the rows.
You should not create both a temporary table and a recursive stored procedure that accepts an array as a parameter and performs a single `INSERT`. There is no need to create a recursive stored procedure. A recursive stored procedure is a stored procedure that calls itself. Stored procedures calls can be nested up to 32 levels. In addition, you cannot pass an array as an input parameter to a stored procedure.

QUESTION 96
You manage a database on an instance of SQL Server 2008 for a large sales organization. You have a **Contact** table defined with the following statement:

```
CREATE TABLE Contact(ContactID int PRIMARY KEY,ContactType nvarchar(10),FirstName
nvarchar(30) NOT NULL,LastName nvarchar(50) NOT NULL,Territory nvarchar(20),
Region nvarchar(10),RepID int,InitContact datetime DEFAULT GETDATE());
```

You have a stored procedure that accepts a territory and region as input parameters and queries the **Contact** table to return contact information as a **table** data type. Users in the marketing department frequently use the stored procedure to obtain information about contacts for a specified territory and region.
Each region contains multiple territories. There are generally a small number of contacts within each territory, but each region contains many contacts.
The stored procedure is performing poorly, and you want to optimize its performance.
Which action should you take?

A. Create a view on the **Contact** table and modify the stored procedure to use the view.
B. Create a nonclustered composite index on the **Territory** and **Region** columns.
C. Create two nonclustered indexes, one on the **Territory** column and one on the **Region** column.
D. Re-create the stored procedure and include the `WITH SCHEMABINDING` clause.

Answer: B
Section: (none)

Explanation/Reference:
To optimize the query in the stored procedure, you should create a composite index on the two columns. In this scenario, each region contains many contacts, and each territory contains only a few. Because there are fewer **Region** values than **Territory** values, you should specify the **Territory** column first when creating the index. With the composite index, SQL Server will find the rows that match the territory and region criteria in the `WHERE` clause with a single index search. The search will be performed only by reading the index pages, and the data pages will be accessed only once to retrieve the result set. You might create the index using the following statement:

```
CREATE INDEX IX_Contact_TerritoryRegion ON Contact (Territory, Region);
```

Using this index would generally offer better performance than if you had created the index with the **Region** column first, because the **Territory**column contains more unique values.
You should not create a view on the **Contact** table and modify the stored procedure to use the view. Using a view would not improve performance. The stored procedure would still have to query the view, and the view would access the underlying table.
You should not create two nonclustered indexes, one on the **Territory** column and one on the **Region** column. In this scenario, a single composite nonclustered index would provide better performance. Although SQL Server can use more than one index on a table, it is unlikely that the index on the **Region** column would be

used in this scenario because of the low uniqueness of the column's values.
You should not re-create the stored procedure including the `WITH SCHEMABINDING` clause because the `WITH SCHEMABINDING` clause is not allowed for a stored procedure. In addition, it has no affect on performance. Attempting to create a stored procedure including `WITH SCHEMABINDING` will generate an error similar to the following:

```
Msg 487, Level 16, State 1, Procedure p, Line 3An invalid option was specified
for the statement "CREATE/ALTER PROCEDURE".
```

Schema binding is used when creating a view or function. You can include the `WITH SCHEMABINDING` clause in a `CREATE VIEW` statement to ensure that no base tables on which the view is based are dropped or modified in a way that might affect the view's definition. To drop base tables or make such modifications, a user would first need to drop the view, alter the view omitting `SCHEMABINDING`, or alter the view to remove any unwanted dependencies. You can also include the `WITH SCHEMABINDING` clause when creating a function to ensure base tables are not changed in such a way that would render the function unusable.

QUESTION 97
There is a query that uses SUM, AVG and MAX. To improve performance:

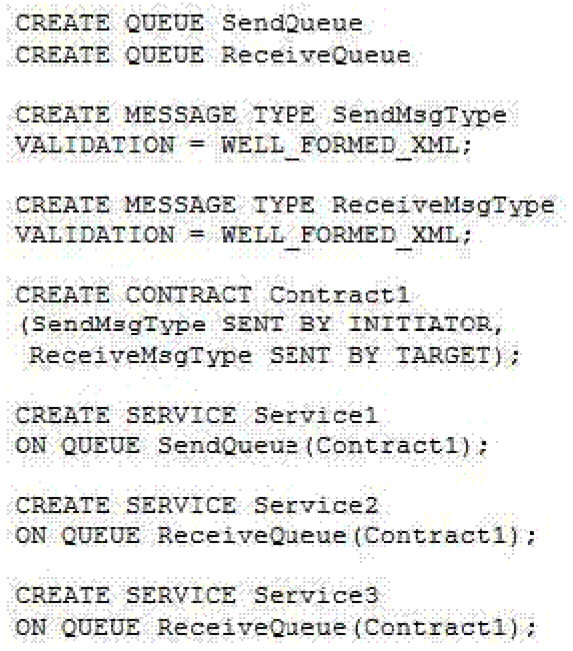
A. create a VIEW that implemented the aggregates as computed columns
B. create an indexed VIEW that implements the aggregates as computed columns
C. create computed columns for aggregates on a per row basis
D. create persisted computed columns for aggregates on a per row basis

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 98
You are a database developer on an instance of SQL Server 2008. You are creating a Service Broker application. You execute the Transact-SQL shown in the exhibit. (Click the **Exhibit(s)** button.) In your application, **Service1** needs to be able to communicate with the other two services. When messages are read from **ReceiveQueue**,
messages delivered to **Service3** should be read first. Which action should you take?

Exhibit:



```
CREATE QUEUE SendQueue
CREATE QUEUE ReceiveQueue

CREATE MESSAGE TYPE SendMsgType
VALIDATION = WELL_FORMED_XML;

CREATE MESSAGE TYPE ReceiveMsgType
VALIDATION = WELL_FORMED_XML;

CREATE CONTRACT Contract1
(SendMsgType SENT BY INITIATOR,
 ReceiveMsgType SENT BY TARGET);

CREATE SERVICE Service1
ON QUEUE SendQueue(Contract1);

CREATE SERVICE Service2
ON QUEUE ReceiveQueue(Contract1);

CREATE SERVICE Service3
ON QUEUE ReceiveQueue(Contract1);
```

A. Create an additional message type and assign it a higher priority.
B. Implement conversation priorities using the `CREATE BROKER PRIORITY` statement.
C. Create separate physical queues for the **Service2** and **Service3** services.
D. Use MSMQ instead of Service Broker because the desired functionality is not supported.

Answer: B
Section: (none)

Explanation/Reference:
By default, messages are sent and received in order. Service Broker allows you to override the order in which messages are sent and received by defining conversation priorities. You can use the `CREATE BROKER PRIORITY` statement to assign different numeric priority levels to different conversations. The syntax of the `CREATE BROKER PRIORITY` statement is as follows:

```
CREATE BROKER PRIORITY priorityname FOR CONVERSATION [SET ([CONTRACT_NAME =
{contractname | ANY}][[,] LOCAL_SERVICE_NAME = {localservicename | ANY}][[,]
REMOTE_SERVICE_NAME = {'remoteservicename' | ANY}][[,] PRIORITY_LEVEL =
{priorityvalue | DEFAULT}])];
```

The `priorityname` specifies the conversation priority's name. The `priorityvalue` specifies an integer between 1 and 10 representing the priority value of the conversation. Conversations with a higher priority value have higher priority. The default `priorityvalue` is 5. The `SET` clause specifies the information that Service Broker uses to decide if a conversation should be prioritized. The `SET` clause may contain one or more of the following criteria:
`CONTRACT_NAME`: The name of the contract used in the conversation that was specified in the `ON CONTRACT` clause of the `BEGIN DIALOG` statement. This criterion allows you to prioritize conversations that use different contracts.
`LOCAL_SERVICE_NAME`: The local service name criterion used to prioritize conversations.
`REMOTE_SERVICE_NAME`: The remote service name criterion used to prioritize conversations.
`PRIORITY_LEVEL`: The integer value representing the conversation priority

Each criterion in the `SET` clause defaults to `ANY` if not specified. When an application sends or receives a message, Service Broker examines the criteria to determine the conversation priority. If Service Broker determines that the conversation meets more than one criterion, it will assign the conversation priority based on the best match.

In this scenario, the **Service2** and **Service3** services share the same queue. However, you want to prioritize the conversations so that when reading messages from **ReceiveQueue**, messages delivered to **Service3** are read first. To accomplish this, you can create two different conversation priorities. For example, you could use the following Transact-SQL to assign a higher priority to **Service3**'s conversations:

```
CREATE BROKER PRIORITY LowPriorityFOR CONVERSATION    SET (CONTRACT_NAME =
Contract1,
         LOCAL_SERVICE_NAME = Service2,         REMOTE_SERVICE_NAME =
N'Service1',         PRIORITY_LEVEL = 1);
  CREATE BROKER PRIORITY HighPriorityFOR CONVERSATION    SET (CONTRACT_NAME =
Contract1,
         LOCAL_SERVICE_NAME = Service3,         REMOTE_SERVICE_NAME =
N'Service1',         PRIORITY_LEVEL = 10);
```

An application could then send a message from **Service1** to **Service2** using **Contract1** and another message from **Service1** to **Service3**. Using the conversation priorities defined by these statements, Service Broker will receive the higher-priority message sent to **Service3** first. Without conversation priorities, the messages would be retrieved from **ReceiveQueue** in the order they were originally sent. You should note that if the conversation priority is bi-directional, separate priorities for each direction must be established.
To determine the message with the highest priority without actually receiving it, you can use the `GET CONVERSATION GROUP` statement. You should also note that although conversations for received messages can implement priorities by default, the `HONOR_BROKER_PRIORITY` database option must be set to `ON` to implement conversation priorities for sent messages.
You should not create an additional message type and assign it a higher priority because Service Broker does not prioritize messages based on the message type.
You should not create separate physical queues for the **Service2** and **Service3** services. In this scenario, it is not necessary to create separate queues for the services. Both services can use the same physical message queue with the conversations for each prioritized differently.
You should not use MSMQ instead of Service Broker because the desired functionality is not supported. Service Broker does support the required functionality with conversation priorities.

QUESTION 99
You are a database developer on an instance of SQL Server 2008. Your **Inventory** database contains several tables that contain information related to historical inventory transactions.
You have created a Transact-SQL stored procedure that will accept a component type, query the tables in the **Inventory** database, and create another table containing all finished good products that have used the specified component. Tables referenced in the stored procedure are owned by different database users.
You want to allow a user, **InvUser**, to call the stored procedure using Transact-SQL, but you do not want to grant the user access to the underlying tables.

Which action should you take?

A. Create an application role and grant the role the needed permissions to the stored procedure. Grant **InvUser** membership in the application role.
B. Create a proxy user **User1**, grant **User1** the needed permissions, and re-create the stored procedure including an `EXECUTE AS User1`
clause. Grant **InvUser** permission to execute the stored procedure.
C. Create a certificate and associate the certificate with **InvUser**. Sign the procedure with the certificate.
D. Grant **InvUser** permission to execute the stored procedure.

Answer: B
Section: (none)

Explanation/Reference:
You should create a proxy user **User1**, grant **User1** the needed permissions, and re-create the stored procedure including an `EXECUTE AS User1` clause. Then, you should grant **InvUser** permission to execute the stored procedure. In this scenario, you need the stored procedure to run under the security context of a

specific user with elevated privileges who has access to objects owned by multiple database users. You want **InvUser** to be able to call the stored procedure without granting **InvUser** permissions on the underlying tables. An ownership chain is created when a database object, such as stored procedure, accesses another database object, such as an underlying table. If a user has access to the stored procedure and the underlying table has the same owner as the stored procedure, then explicit permission to the underlying table is not required. SQL Server will only check permissions on the underlying table if it has a different owner. However, you must note that ownership chaining does not occur when the stored procedure contains DDL.
In this scenario, the underlying tables are owned by multiple users, and the stored procedure contains DDL. Therefore, ownership chaining will not occur. To allow **InvUser** access, you can use the `EXECUTE AS` clause to force the stored procedure to execute with additional privileges. When creating a stored procedure, you can include the `EXECUTE AS` clause to specify the security context under which the stored procedure should execute. You can specify the following values in the `EXECUTE AS` clause:
`SELF`: Specifies the stored procedure executes under the security context of the current user.
`OWNER`: Specifies the stored procedure executes under the security context of the user that owns it.
`CALLER`: Specifies the stored procedure executes under the security context of the user calling it. To execute the stored procedure successfully, the user calling the stored procedure would require permissions on both the stored procedure and on any underlying database objects referenced by the stored procedure.
`user_name`: Specifies the stored procedure executes under the security context of the specified user, regardless of which user called the stored procedure.

By default, a stored procedure executes under the security context of the caller. However, in this scenario you can create a proxy user with additional permissions, **User1**, and force the stored procedure to run under **User1** 's security context by creating the stored procedure with the `EXECUTE AS User1` clause. Then, you can grant **InvUser** permission to execute the stored procedure. When **InvUser** executes the stored procedure, it will run with **User1**'s permissions.
You should not create an application role with the required permissions and grant **InvUser** membership in the application role because application roles are not granted directly to users. Application roles can allow users to use the database only through a custom application. The users are not directly assigned permissions, but rather are allowed permissions through the application role only. Using application roles prevents application users from performing any tasks outside the custom application. To implement an application role, you create an application role with a password using the `CREATE APPLICATION ROLE` statement. An application role does not contain users, only a password. Then, you assign the permissions required by the application to the application role. Finally, you code the application to authenticate users, and activate the appropriate application role using the **sp_setapprole** system stored procedure. After the application activates the role, the user will have only the role's permissions, and any other database permissions granted to the user are disregarded.
You should not create a certificate and sign the procedure with the certificate. You can use certificates to provide permissions. To do so, you would create a certificate, associate the certificate with a specific user, and grant the user the needed permissions. Then, you would sign the stored procedure with the certificate. When the procedure is called, it will execute with the certificate user's permissions. You use the `CREATE CERTIFICATE` statement to create a certificate. To create a user associated with the certificate, you can use the `CREATE USER` statement and include the `FROM CERTIFICATE` clause. To sign the procedure, you can use the `ADD SIGNATURE` statement, specifying the stored procedure name, certificate, and certificate password. You should note that each time you alter the stored procedure you must re-sign the stored procedure.
You should not grant **InvUser** permission to execute the stored procedure. In the given scenario, the stored procedure would run by default the caller's security context. Ownership chaining would not occur, and access to the underlying tables would be denied because the underlying tables have different owners.

QUESTION 100
You are a database administrator on an instance of SQL Server 2008. You maintain a database named **Prod** that contains information on products manufactured by your company, including product assemblies, bills of materials, work order information, and related component and finished-goods inventory information. Employees in the inventory department use a custom ASP.NET application to manage the details about products that are received in inventory and finished goods that are transferred from inventory locations to fill and ship customer orders. Employees in the assembly department access bills of materials, update

component inventories, and update work order status in the database using a Windows Forms application. You want to minimize administrative effort, ensure that employees can only access the necessary data in the **Prod** database, and prevent users from directly querying database tables.

Which action should you take?

A. Create an application role for each department. Add each employee to the appropriate application role.
B. Create a view to be used by each application. Grant each application permission to use the appropriate view.
C. Create an application role for each department, assign each departmental application role the required permissions, and code each application to authenticate users and activate the appropriate application role
D. Create a database role for each department, assign the required object permissions to each database role, and add each employee to the appropriate database role.

Answer: C
Section: (none)

Explanation/Reference:
You should create an application role for each department, assign each departmental application role the required permissions, and code each application to authenticate users and activate the appropriate application role. Application roles allow users to use the database only through a custom application. The users are not directly assigned permissions, but rather are allowed the permissions granted to the application role. Using application roles prevents users from directly accessing database objects. To implement an application role, you create an application role with a password using the `CREATE APPLICATION ROLE` statement. An application role does not contain users, only a password. Then, you assign the permissions required by the application to the application role. Finally, you code the application to authenticate users and activate the appropriate application role using the **sp_setapprole** system stored procedure. If users have been granted other database permissions, those permissions are disregarded. In this scenario, you would create an application role for each custom application and grant each application role only the necessary permissions. Then, you would have each application authenticate users and activate the appropriate application role.
You should not create a database role for each department, assign the required object permissions to each database role, and add each employee to the appropriate database role. Database roles allow users to directly access the database, not to access the database via a custom application. A database role would allow users to directly access database objects, which should not be allowed in this scenario.
You should not create an application role for each department and add each employee to the appropriate application role. Users cannot be assigned to application roles.
You should not create a view to be used by each application and grant each application permission to use the appropriate view because this would allow users direct database access.

QUESTION 101
You are a SQL Server 2008 developer. You create an online transaction processing (OLTP) database by using SQL Server 2008 in an enterprise environment. The database contains a table named SalesDetails. Each record in the table contains data in any one of the following pairs of nullable columns:
- InternetSalesTargets and InternetSales
- ResellerSalesTargets and ResellerSales
- ForeignSalesTargets and ForeignSales

The table also contains three NOT NULL key columns. A large number of records are inserted on a daily basis into the SalesDetails table. Summary reports are generated from the SalesDetails table. Each report is based on aggregated data from any one of the pairs of nullable columns. You need to design a view or views to meet the following requirements:
- The SalesDetails table cannot be directly modified.
- The performance of the reports is maximized.
- The amount of storage space for each report is minimized.

What should you do?

A. Create an indexed view from the SalesDetails table that contains aggregated data of all the columnsrequired by all the reports.
B. Create multiple indexed views from the SalesDetails table so that each view contains aggregated data of only the columns required by the respective report.
C. Create multiple Report tables from the SalesDetails table so that each Report table contains aggregated data of only the columns required by the respective report. Create views on top of each of the Report tables.
D. Perform a quick transfer of aggregated new records to a staging table at the end of each month. Create an indexed view from the staging table that contains aggregated data of all the columns requiredby all the reports.

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 102
You are a database developer. You plan to design a database solution by using SQL Server 2008. A database contains a view that has the following features:
·
It contains a WHERE clause that filters specific records.
·
It allows data updates.
You need to prevent data modifications that do not conform to the WHERE clause. You want to achieve this goal by using minimum effort.
What should you do?

A. Create an INSTEAD OF trigger on the view.
B. Create a unique clustered index on the view.
C. Alter the view by adding the WITH CHECK OPTION clause.
D. Alter the view by adding the WITH SCHEMABINDING clause.

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 103
You maintain an **Inventory** database on an instance of SQL Server 2008. You have a large number of XML documents that you receive from a third-party processor each week.
At the end of each week, you want to shred the XML into relational data and populate existing tables in your database. After this process is complete, you want to archive the XML documents to folders on the file system. You want to accomplish this with the least development effort. Which action should you take?

A. Create an SSIS package to perform the needed tasks.
B. Use XQuery and the **query()** method.
C. Create a temporary table containing an **xml** data type column and use the `OPENXML` function.
D. Use `FOR XML AUTO` with the `XMLDATA` option.

Answer: A

Section: (none)

Explanation/Reference:
In this scenario, you need to shred multiple incoming XML documents into relational data and use it to populate existing database tables. You also need to repeat this process each week as new documents are received, and archive the XML documents to folders on the file system after they are processed. For this scenario, the best choice would be to use an SSIS package. SSIS allows you to extract, transform, and load data into data sources from a variety of formats, including XML documents. You would be able to manipulate the incoming XML as needed and add it to your existing tables, and then archive the data to the file system. You could then schedule the process to run weekly. This solution would minimize the development effort.
You should not use XQuery and the **query()** method. The **query()** method is used to retrieve nodes from existing **xml** columns or variables. In this scenario, you need to load the data and shred it as it is loaded, and you need to repeat this weekly. Therefore, using an SSIS package to perform the task would be better.
You should not create a temporary table containing an **xml** data type column and use the `OPENXML` function because this would require more coding than necessary. You would also need to implement a mechanism for archiving the XML documents and a mechanism to perform the tasks weekly. `OPENXML` is a rowset provider function that creates a relational view of data contained in an XML document. This function can be used to shred XML into relational data by including the function in a `SELECT` statement as if it were a table or view. To use `OPENXML`, you first call the **sp_xml_preparedocument** system stored procedure to obtain a handle to the XML instance. Then, you can extract data from the XML document into relational format. After shredding the XML, you call the **sp_xml_removedocument** stored procedure to remove the document handle and free up resources.
You should not use `FOR XML AUTO` with the `XMLDATA` option because the `FOR XML` clause is used with a `SELECT` statement to output relational data in XML format. In previous versions of SQL Server, the `XMLDATA` option allowed you to include an XML-Data Reduced (XDR) schema in the generated output. In situations where this might be applicable, you should use the `XMLSCHEMA` option instead, because the `XMLDATA` option has been deprecated.

QUESTION 104
You are a database developer on an instance of SQL Server 2008. You are creating an application that will use Service Broker objects to send and receive messages. You want to create a service named **BasicService** that will allow both conversation participants to send and receive messages of the `DEFAULT` type.
Which Transact-SQL statement must you execute before creating the **BasicService** service?

A. `CREATE CONTRACT`
B. `CREATE BROKER PRIORITY`
C. `CREATE QUEUE`
D. `CREATE ROUTE`

Answer: C
Section: (none)

Explanation/Reference:
A queue stores the incoming message for the service. Each service must be associated with a queue. Therefore, you must create a queue using the `CREATE QUEUE` statement before creating a service. After creating the queue, you can execute the `CREATE SERVICE` statement. The basic syntax of the `CREATE SERVICE` statement is as follows:
```
CREATE SERVICE service_name [AUTHORIZATION owner_name]ON QUEUE [schema_name.]
queue_name [(contract_name | [DEFAULT])[,...n ]];
```
The `AUTHORIZATION` clause identifies the owner of the service. The `ON QUEUE` clause specifies the queue that the service will use to store messages and any contracts that define the allowed message types. You must specify a value for the `ON QUEUE` clause to associate the service with a message queue. Service Broker uses the service name to identify the queue used for the service. When creating a service, specifying one or

more contracts is optional. If you omit the contract name(s), the service can only start a conversation. If you use the `DEFAULT` contract, both conversation participants will be able to send and receive messages of the `DEFAULT` type.
The basic steps to create a Service Broker application that sends and receives messages between two services are as follows:
Create message types using `CREATE MESSAGE TYPE` statements.
Create contracts using `CREATE CONTRACT` statements.
Create the queue to store messages using the `CREATE QUEUE` statement.
Create the services using `CREATE SERVICE` statements.
Begin a conversation between the two services using the `BEGIN DIALOG CONVERSATION` statement.
Send and receive messages using the `SEND` and `RECEIVE` statements, respectively.

You do not have to execute the `CREATE CONTRACT` statement. You use the `CREATE CONTRACT` statement to create a contract. A contract defines the message types that are allowed and the conversation participants that can send messages. However, you can create a service without specifying a contract or using the `DEFAULT` contract. For example, the following statement creates a contract that allows the conversation initiator to send messages of the **BasicMessageType** type:

```
CREATE CONTRACT BasicContract (BasicMessageType SENT BY INITIATOR);
```

You do not have to execute the `CREATE BROKER PRIORITY` statement. The `CREATE BROKER PRIORITY` statement is used to create a conversation priority level that Server Broker uses to prioritize conversations. The statement includes the criteria that Service Broker uses to determine the conversation's priority. When you have conversations prioritized, you can use the `GET CONVERSATION GROUP` statement to determine the conversation group that has the highest priority.
You do not have to execute the `CREATE ROUTE` statement. The `CREATE ROUTE` statement creates a route that specifies how Service Broker will locate services on other computers or in other databases.

QUESTION 105
You are a database administrator for NuTex Corporation. Your company maintains survey data in the **Research** database residing on an instance of SQL Server 2008. You are creating a Service Broker application to send and receive messages between two services, **//NuTex.com/SurveyInput** and **//NuTex.com/Respondent**.
You want to receive survey-related information from another organization that maintains data on a different database, **Prod**. The survey data sent from the **Prod** database is in XML format. At periodic intervals, the **Research** database will initiate a conversation to request survey response data.
You have created the message type to validate the incoming survey data using the following statement:

```
CREATE MESSAGE TYPE [//NuTex.com/Research/SurveyData] VALIDATION =
WELL_FORMED_XML;
```

Which statement should you use to create the contract to be used in conversations?

A.
```
CREATE CONTRACT [//NuTex.com/Research/SurveyData/Contract1]
([//NuTex.com/Research/SurveyData] SENT BY TARGET);
```
B.
```
CREATE CONTRACT [//NuTex.com/Research/SurveyData/Contract1]
([//NuTex.com/Research/SurveyData] SENT BY ANY);
```
C.
```
CREATE CONTRACT [//NuTex.com/Research/SurveyData/Contract1]
(SurveyData SENT BY INITIATOR);
```
D.
```
CREATE CONTRACT [//NuTex.com/Research/SurveyData/Contract1]
([DEFAULT] SENT BY ANY);
```

Answer: B
Section: (none)

Explanation/Reference:
You should use the following statement to create the contract:

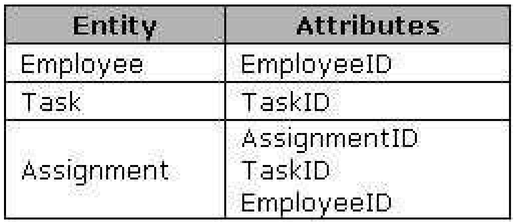
```
CREATE CONTRACT [//NuTex.com/Research/SurveyData/Contract1]([//NuTex.com/
Research/SurveyData] SENT BY ANY);
```

In a Service Broker conversation, a contract defines the types of messages allowed and the conversation participants that can send these messages. In this scenario, you created a message type named **//NuTex.com/Research/SurveyData**. You can then use the `CREATE CONTRACT` statement to create the contract to be used in conversations. The syntax of the `CREATE CONTRACT` statement is as follows:

```
CREATE CONTRACT contract_name [AUTHORIZATION owner_name]({{message_type |
[DEFAULT]}SENT BY {INITIATOR | TARGET | ANY} [,...n]);
```

In this scenario, the **Research** database is the initiator and receives messages from the **Prod** database. To accomplish this, you must indicate that the conversation initiator can send messages by specifying either `SENT BY INITIATOR` or `SENT BY ANY` in the `CREATE CONTRACT` statement for the message type.
You should not use the statement that includes the `SENT BY TARGET` clause because this statement will generate an error. Specifying the `SENT BY TARGET` clause indicates that only the target in the conversation can send messages of the specified message type. Each contract must contain at least one message specifying `SENT BY INITIATOR` or `SENT BY ANY`, or an error occurs.
You should not use the statement that specifies `SurveyData` as the message type because this statement will generate a syntax error. In this scenario, the name of the message type is **//NuTex.com/Research/SurveyData**.
You should not use the statement that specifies `[DEFAULT]` as the message type because the `[DEFAULT]` message type has validation set to `NONE`, indicating the data is not validated. In this scenario, the data is in XML format and requires the validation as specified in the message type you created. When you create a message type, you can specify one of the following types of validation:
`EMPTY`: Indicates that the message body will be `NULL`.
`NONE`: Indicates that validation will not be performed. This type of validation is used for binary data.
`WELL_FORMED_XML`: Indicates that the message body will contain well-formed XML data.
`VALID_XML WITH SCHEMA COLLECTION schema_collection_name`: Indicates that the message body will contain XML data that will be validated against a predefined XML schema in the specified schema collection.

QUESTION 106
You are a database developer. You plan to design a database solution by using SQL Server 2008. You are creating a database to support the office manager. Your database model has the following structure.



Entity	Attributes
Employee	EmployeeID
Task	TaskID
Assignment	AssignmentID TaskID EmployeeID

The database design has the following business requirements:
- An employee can be assigned more than one task.
- Upon completion, the task is deleted.
- When a task is deleted, the associated assignment is deleted.
- When an employee is no longer available to complete a task, the employee link to the assignment is replaced with a NULL value.

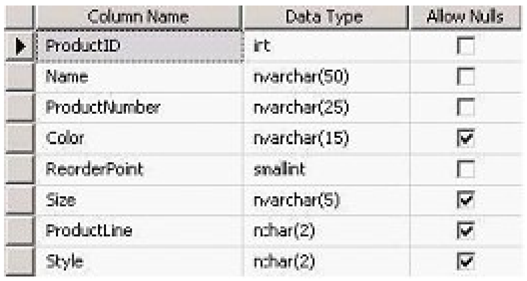
You need to implement the business requirements to maintain data integrity.What should you do?

A. Create DDL INSERT triggers on the Employee, Task, and Assignment entities.
B. Create CHECK constraints on the TaskID and EmployeeID attributes in the Assignment entity.
C. Create Foreign Keys constraints on the TaskID and EmployeeID attributes in the Assignment entity.
D. Create Foreign Keys constraints on the TaskID and EmployeeID attributes in the Task and Employeeentities respectively.Reference the Assignment entity, and specify the appropriate On Delete action.

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 107
You are a developer on an instance of SQL Server 2008. You maintain the **Prod** database, which includes the **ProductDetails** table defined as follows:



Column Name	Data Type	Allow Nulls
ProductID	int	☐
Name	nvarchar(50)	☐
ProductNumber	nvarchar(25)	☐
Color	nvarchar(15)	☑
ReorderPoint	smallint	☐
Size	nvarchar(5)	☑
ProductLine	nchar(2)	☑
Style	nchar(2)	☑

You want to create a user-defined scalar function to return the **ReorderPoint** value for a specific product. If the **ProductID** value passed to the function is between 100 and 500, the function should return the actual **ReorderPoint** value. For other products, a reorder point of zero should be
returned.
Which statement should you use to create the function?

A.
```
CREATE FUNCTION dbo.udf_get_reorder_point(@v_prodid int)
RETURNS int
AS
BEGIN
DECLARE @v_reorderpt int;
SELECT @v_reorderpt = ReorderPoint
 FROM ProductDetails
 WHERE ProductID = @v_prodid
 IF (@v_prodid BETWEEN 100 AND 500)
  RETURN @v_reorderpt
else
 RETURN 0
END;
```

B.
```
CREATE FUNCTION dbo.udf_get_reorder_point(@v_prodid int)
RETURNS int
AS
BEGIN
DECLARE @v_reorderpt int;
IF (@v_prodid BETWEEN 100 AND 500)
BEGIN
 SELECT @v_reorderpt = ReorderPoint FROM ProductDetails WHERE ProductID =
@v_prodid
 RETURN @v_reorderptEND
ELSE
 RETURN 0
END;
```

C.
```
CREATE FUNCTION dbo.udf_get_reorder_point(@v_prodid int)
AS
BEGIN
DECLARE @v_reorderpt int;
SELECT @v_reorderpt = ReorderPoint FROM ProductDetails WHERE ProductID =
@v_prodid
IF (@v_prodid BETWEEN 100 AND 500)RETURN @v_reorderpt
RETURN 0
END;
```
D.
```
CREATE FUNCTION dbo.udf_get_reorder_point(@v_prodid int)
RETURNS int
AS
BEGIN
DECLARE @v_reorderpt int;
IF (@v_prodid BETWEEN 100 AND 500)
 SELECT @v_reorderpt = ReorderPoint FROM ProductDetails WHERE ProductID =
@v_prodid
ELSE SET @v_reorderpt = 0
END;
```

Answer: A
Section: (none)

Explanation/Reference:
You should use the following statement to create the function:
```
CREATE FUNCTION dbo.udf_get_reorder_point(@v_prodid int)RETURNS int AS BEGIN
DECLARE @v_reorderpt int;SELECT @v_reorderpt = ReorderPoint FROM ProductDetails
WHERE ProductID = @v_prodidIF (@v_prodid BETWEEN 100 AND 500)RETURN @v_reorderpt
RETURN 0 END;
```
In this scenario, you wanted to create a user-defined scalar function to return the **ReorderPoint** value for a specific product, but you wanted the function to return a zero value if the **ProductID** value was not between 100 and 500. A scalar function returns a single scalar value. You can create a user-defined function using the `CREATE FUNCTION` statement. The basic syntax of the `CREATE FUNCTION` statement when creating a scalar function is as follows:
```
CREATE FUNCTION [schema_name.]function_name ([{@parm_name [AS][parmtype_schema.]
parm_data_type [=default] [READONLY]} [,...n]])RETURNS return_type [WITH
function_opt [,...n]][AS]
BEGIN
function_body RETURN scalar_expression END;
```
You can pass no parameters or multiple parameters into the function using the parameter list. You specify the `RETURNS` clause to indicate the data type of the value that the function returns. Then, you include Transact-SQL statements within the body of the function, and use the `RETURN` statement to return the value. In this scenario, the statement accepts a parameter that identifies a product in the **ProductDetails** table, and declares and sets the value of a variable to return the **ReorderPoint** value. The `IF` statement checks the **ProductID** value that was passed to the function. If the **ProductID** value is in the desired range, the function executes the first `RETURN` statement and returns the actual **ReorderPoint** value. However, if the **ProductID** is not in the desired range, the second `RETURN` statement executes and returns a zero value to the caller. After you create the function, you could call the function from other Transact-SQL code. For example, you could use the following statement to return the **ReorderPoint** value for the product with a **ProductID** value of 100:
```
SELECT dbo.udf_get_reorder_point(100);
```
You should not use the `CREATE FUNCTION` statement that includes an `ELSE` because this statement will return the following error:
```
Msg 455, Level 16, State 2, Procedure udf_get_reorder_point, Line 16The last
statement included within a function must be a return statement.
```
Even though the last `RETURN` statement may be the last executed, it is not considered the last statement in the function and will generate an error.
You should not use the `CREATE FUNCTION` statement that omits the `RETURNS` clause because a `RETURNS` clause is required to identify the data type returned. The following statement will generate a syntax error:

```
CREATE FUNCTION dbo.udf_get_reorder_point(@v_prodid int)AS BEGIN DECLARE
@v_reorderpt int;SELECT @v_reorderpt = ReorderPoint FROM ProductDetails WHERE
ProductID = @v_prodidIF (@v_prodid BETWEEN 100 AND 500)RETURN @v_reorderpt
RETURN 0 END;
```

You should not use the `CREATE FUNCTION` statement that does not include a `RETURN` statement. Each function must contain a `RETURN` statement to return the value, and the statement must be the last statement in the function. The following statement will generate an error as shown:

```
CREATE FUNCTION dbo.udf_get_reorder_point(@v_prodid int)RETURNS int AS
BEGIN DECLARE @v_reorderpt int;IF (@v_prodid BETWEEN 100 AND 500)SELECT
@v_reorderpt = ReorderPoint FROM ProductDetails WHERE ProductID = @v_prodidELSE
SET @v_reorderpt = 0END;
Msg 455, Level 16, State 2, Procedure udf_get_reorder_point, Line 7The last
statement included within a function must be a return statement.
```

QUESTION 108

You manage a large database on an instance of SQL Server 2008. You have a table named **TrxHistory** that contains millions of rows and multiple columns, including a **TrxDate** column and a **TrxStatus** column. Most of the rows have a **TrxStatus** value of 'D', and most of your queries process only these rows for specified date ranges.
You decide to create a filtered index to improve query performance.
Which statement will successfully create a filtered index?

A. `CREATE NONCLUSTERED INDEX FI_TrxStatusD ON TrxHistory (TrxDate)WITH (IGNORE_DUP_KEY = OFF)WHERE TrxStatus = 'D';`

B. `CREATE NONCLUSTERED INDEX FI_TrxStatusD ON TrxHistory (TrxDate, TrxStatus = 'D');`

C. `CREATE NONCLUSTERED INDEX FI_TrxStatusD ON TrxHistory (TrxDate)WHERE TrxStatus = 'D';`

D. `CREATE NONCLUSTERED INDEX FI_TrxStatusD ON TrxHistory (TrxDate) USING TrxStatus = 'D';`

Answer: C
Section: (none)

Explanation/Reference:
A filtered index is a nonclustered index that is defined with a specific `WHERE` clause to optimize the index for specific queries. The index uses the `WHERE` clause condition to index only specific rows in the table. Using a filtered index can improve performance in many situations and reduce the space required for the index. The given statement creates an index named **FI_TrxStatusD** on the **TrxDate** column of the **TrxHistory** table, but only includes rows with a **TrxStatus** value of D. You should note that the `WHERE` clause cannot reference certain types of columns, such as a computed column, a user-defined type, or a **geography**, **geometry**, or **hierarchyid** data type.
The statement that uses the `IGNORE_DUP_KEY` option when creating the index is incorrect because the `IGNORE_DUP_KEY` option cannot be specified with filtered indexes.
The statements that specify `ON TrxHistory (TrxDate, TrxStatus = 'D')` and `ON TrxHistory (TrxDate) USING TrxStatus = 'D'` are incorrect because these statements will each generate a syntax error. To create a filtered index, a `WHERE` clause condition must be used.

QUESTION 109

You are a database developer. You plan to design a database solution by using SQL Server 2008.
The database application has a table named Transactions that contains millions of rows. The table has multiple columns that include transaction_id and transaction_date. There is a clustered index on the transaction_id column. There is a nonclustered index on the transaction_date column.
You discover that the following query takes a long time to execute. SELECT transaction_id, transaction_date, transaction_notes FROM transactions

WHERE transaction_type_id = 'FXO'
AND transaction_date between @start_date and @end_date The summary of the execution plan is as shown in the following code segment.
|--Filter(WHERE:([transaction_type_id]='FXO')
|--Nested Loops(Inner Join)
|--Index Seek(OBJECT:([transactions]. [nc_transactions_transaction_date]) |--Clustered Index Seek(OBJECT:([transactions]. [PK_transactions_transaction_id]) You need to ensure that the query retrieves data in minimum possible time.
What should you do?

A. Create a nonclustered index on the transaction_type_id column.

B. Create a nonclustered index on the transaction_date and transaction_type_id columns.
C. Create a nonclustered index on the transaction_date column and include the transaction_type_id and transaction_notes columns.
D. Create a nonclustered index on the transaction_date and transaction_type_id columns and include the transaction_notes column.

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 110
You have a legacy application. You do not have access to the application source code. The application has a large denormalized table that contains 100 columns. The application uses stored procedures and views to perform all data manipulation language (DML) activities on the table. You need to optimize the performance of the application to meet the following requirement:
- Reduce I/O
- Minimize the storage requirements
- Optimize insert, update, and delete operations

What should you do?

A. Create nonclustered indexes on all columns in the table.
B. Create new stored procedures that use the existing views.
C. Create new views. Perform DML activities against the views
D. Create smaller tables. Update the views and stored procedures.

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 111
You need to create an application that will represent the relationship between managers and employees. You must achieve this goal by using the minimum amount of tables.
What should you do?

A. Create one table that contains the hierarchyid data type.

B. Create one table that contains the uniqueidentifer data type.

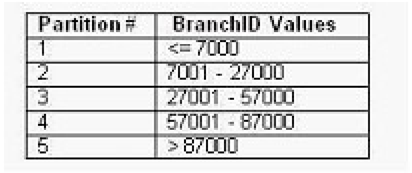
C. Create two tables. Establish a foreign key relationship between the tables.

D. Create two tables. Create a trigger that maintains the relationship between the two tables.

Answer: A
Section: (none)

Explanation/Reference:

QUESTION 112
You are a database developer on an instance of SQL Server 2008. You have over a million rows of sales history data that you plan to import into the **Sales** database. You want to create a **SalesHistory** table to store the data.
Sales history for each branch is identified by the **BranchID** column, which is an **int** data type, and most queries will be executed by branch managers for their specific branch's data. You decide to partition the table to improve query performance.
You want to partition the **SalesHistory** table as follows:



Partition #	BranchID Values
1	<= 7000
2	7001 - 27000
3	27001 - 57000
4	57001 - 87000
5	> 87000

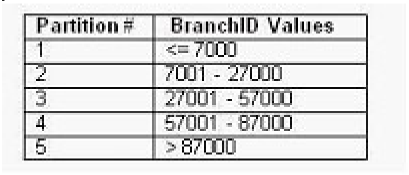
Which Transact-SQL should you use to create the **SalesHistory** table as a partitioned table?

A.
```
CREATE PARTITION FUNCTION MyPF (int)
AS RANGE LEFT FOR VALUES (6999, 26999, 56999, 86999);
CREATE PARTITION SCHEME MyPSAS PARTITION MyPFTO (fg1, fg2, fg3, fg4, fg5);
CREATE TABLE SalesHistory (OrderID int,OrderDate datetime,TerritoryID int,
BranchID int,TaxFlg bit,ShipMethodID int,Amount money)ON MyPS(BranchID);
```
B.
```
CREATE PARTITION FUNCTION MyPF (int)
AS RANGE LEFT FOR VALUES (7000, 27000, 57000, 87000);
CREATE TABLE SalesHistory (OrderID int,OrderDate datetime,TerritoryID int,
BranchID int,TaxFlg bit,ShipMethodID int,Amount money)ON MyPF(BranchID);
```
C.
```
CREATE PARTITION FUNCTION MyPF (int)AS RANGE LEFT FOR VALUES (7000, 27000,
57000, 87000);
CREATE PARTITION SCHEME MyPSAS PARTITION MyPFTO (fg1, fg2, fg3, fg4, fg5);
CREATE TABLE SalesHistory (OrderID int,OrderDate datetime,TerritoryID int,
BranchID int,TaxFlg bit,
ShipMethodID int,Amount money)ON MyPS(BranchID);
```
D.
```
CREATE PARTITION FUNCTION MyPF (int)
AS RANGE LEFT FOR VALUES (7000, 27000, 57000, 87000);
CREATE PARTITION SCHEME MyPSAS PARTITION MyPFTO (fg1, fg2, fg3, fg4, fg5);
CREATE TABLE SalesHistory (OrderID int,OrderDate datetime,TerritoryID int,
BranchID int,TaxFlg bit,ShipMethodID int,Amount money)ON MyPS(OrderDate);
```

Answer: B
Section: (none)

Explanation/Reference:
To create a partitioned a table, you first create a partition function using the `CREATE PARTITION FUNCTION` statement that specifies the number of partitions and how partitioning will occur. A partition function maps the rows of a table or index into partitions based on the specified partition boundary values. The `CREATE PARTITION FUNCTION` statement in this scenario creates a partition function named **MyPF** with a partition

column of the **int** data type. The `FOR VALUES` clause of the `CREATE PARTITION FUNCTION` statement specifies the boundary value of each partition. The `RANGE RIGHT` and `RANGE LEFT` clauses are used to specify how the actual boundary values are handled. `RANGE LEFT` indicates that the boundary value should be stored in the partition on the left side of the boundary value with the boundary values sorted in ascending order from left to right. The `CREATE PARTITION FUNCTION` statement in this scenario, defines five partitions as follows:

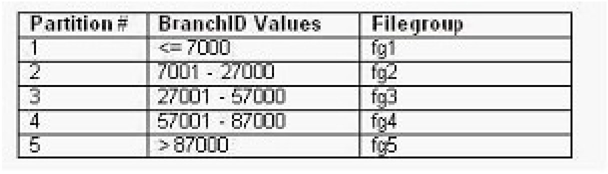


Partition #	BranchID Values
1	<= 7000
2	7001 - 27000
3	27001 - 57000
4	57001 - 87000
5	> 87000

Next, you must create a partition scheme based on the previously created partition function using the `CREATE PARTITION SCHEME` statement. The partition scheme maps the partitions created by the partition function to filegroups. One or more filegroups can be specified in the partition scheme. The `AS PARTITION` clause of the `CREATE PARTITION SCHEME` statement identifies the partition function, and the `TO` clause specifies the filegroups. The complete syntax for the `CREATE PARTITION SCHEME` statement is:

```
CREATE PARTITION SCHEME name_of_partition_scheme AS PARTITION
name_of_partition_function [ALL] TO ({file_group | [PRIMARY]} [,...n] );
```

The partition scheme created in this scenario would map the partitions to file groups as follows:



Partition #	BranchID Values	Filegroup
1	<= 7000	fg1
2	7001 - 27000	fg2
3	27001 - 57000	fg3
4	57001 - 87000	fg4
5	> 87000	fg5

After you have created the partition function and partition scheme, you must create the **SalesHistory** table as a partitioned table using a `CREATE TABLE` statement that includes an `ON` clause. The `ON` clause identifies the partition scheme and the column on which the table will be partitioned. The specified partitioning scheme identifies the partition function that is used. The complete syntax of the `CREATE TABLE` statement to create a partitioned table is as follows:

```
CREATE TABLE table_name (column_def1, column_def2, ...)ON
name_of_partition_scheme (partition_column_name);
```

The arguments used in the statement syntax are as follows:

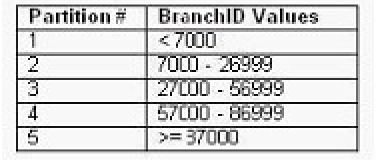
`table_name`: Specifies the name of the table to be created.

`column_defn`: Specifies the details of the column(s) in the table.

`partition_scheme_name`: Specifies the name of the partition scheme that identifies the partition function and the filegroups to which the partitions of the table will be written.

`partition_column_name`: Specifies the name of the column in the table on which the table will be partitioned. The column specified must match the column definition specified in the associated partition function in terms of the data type, length, and precision.

You should not use the Transact-SQL that includes `RANGE LEFT FOR VALUES (6999, 26999, 56999, 86999)` in the `CREATE PARTITION FUNCTION` statement because this will partition the **SalesHistory** table as follows:



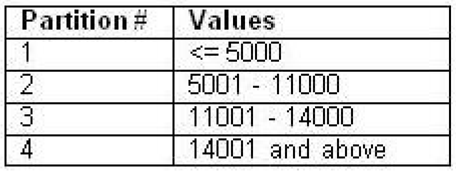
Partition #	BranchID Values
1	< 7000
2	7000 - 26999
3	27000 - 56999
4	57000 - 86999
5	>= 37000

You should not use the Transact-SQL that fails to include a `CREATE PARTITION SCHEME` statement. You must create a partition scheme and specify the partition scheme, not the partition function, in the `ON` clause of the `CREATE TABLE` statement. This code will generate the following error:

```
Msg 1921, Level 16, State 1, Line 4Invalid partition scheme 'MyPF' specified.
```

You should not use the Transact-SQL that specifies `MyPS(OrderDate)` in the `ON` clause of the `CREATE TABLE` statement. When creating a partitioned table, the data type of the column specified in the `ON` clause must be the same as the data type specified in the `CREATE PARTITION FUNCTION` statement, which in this scenario is **int**. This code will generate an error.

QUESTION 113
You are a database developer on an instance of SQL Server 2008. You have a **SalesHistory** table. You want to partition the **SalesHistory** table as
follows based on the value of the **OrderID** column:



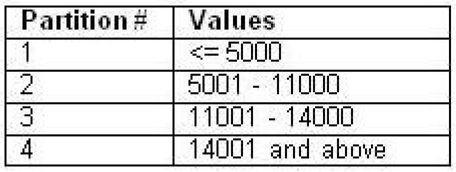
Partition #	Values
1	<= 5000
2	5001 - 11000
3	11001 - 14000
4	14001 and above

Which Transact-SQL statement should you use to create the partition function?

A.
```
CREATE PARTITION FUNCTION Pf1 (int)
AS RANGE RIGHT
FOR VALUES (20000, 40000);
```
B.
```
CREATE PARTITION FUNCTION Pf1 (int)
AS RANGE LEFT
FOR VALUES (20000, 40000);
```
C.
```
CREATE PARTITION FUNCTION Pf1 (int)
AS RANGE RIGHT
FOR VALUES (<20000, <40000, >40000);
```
D.
```
CREATE PARTITION FUNCTION Pf1 (int)
AS RANGE LEFT
FOR VALUES (19999, 20001, 40000);
```

Answer: A
Section: (none)

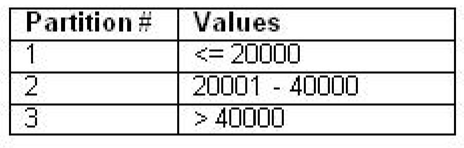
Explanation/Reference:
A partition function maps the rows of a table or index into partitions based on the specified partition boundary values. The `CREATE PARTITION`
`FUNCTION` statement in this scenario creates a partition function named **Pf1** with a partition column of the **int** data type. The `FOR VALUES` clause of
the `CREATE PARTITION FUNCTION` statement specifies the boundary value of each partition. The `RANGE RIGHT` and `RANGE LEFT` clauses
specify how the actual boundary values are handled. `RANGE LEFT` indicates that the boundary value should be stored in the partition on the left
side of the boundary value, with the boundary values sorted in ascending order from left to right. The `CREATE PARTITION FUNCTION` statement in
this scenario defines three partitions as follows:



Partition #	Values
1	<= 5000
2	5001 - 11000
3	11001 - 14000
4	14001 and above

The complete syntax for the `CREATE PARTITION FUNCTION` statement is:
```
CREATE PARTITION FUNCTION name_of_partition_function (type_of_input_parameter)
AS RANGE [LEFT | RIGHT]
FOR VALUES ([boundary_value [,...n]]);
```
You should not use the statement that includes `FOR VALUES (20000, 40000)` but includes the `AS RANGE LEFT` clause. This would place the

boundary value in the partition to the left of the boundary value. This statement would partition the table as follows:



Partition #	Values
1	<= 20000
2	20001 - 40000
3	> 40000

QUESTION 114
You are a database developer on an instance of SQL Server 2008. You are creating a stored procedure to update specific values in the **SalesHistory** table. Your stored procedure will accept two scalar values, query the **SalesHistory** table and various lookup tables, and update another table in the database based on the query's results.
You want the stored procedure to always execute using the security context that was in effect when the stored procedure was created. You also want to ensure that the stored procedure's definition cannot be viewed by users.
Which Transact-SQL should you use to create the stored procedure?

A.
```
CREATE PROCEDURE usp_UpdateSalesHistory
(@id int, @date datetime)WITH EXECUTE AS DevUser, ENCRYPTIONAS BEGIN -- Code to
retrieve data from SalesHistory and update the desired table
END;
```
B.
```
CREATE PROCEDURE usp_UpdateSalesHistory
(@id int, @date datetime)WITH ENCRYPTION, RECOMPILEAS BEGIN -- Code to retrieve
data from SalesHistory and update the desired table
END;
```
C.
```
(@id int, @date datetime)WITH EXECUTE AS OWNER, ENCRYPTIONAS BEGIN -- Code to
retrieve data from SalesHistory and update the desired table
CREATE PROCEDURE usp_UpdateSalesHistory
END;
```
D.
```
CREATE PROCEDURE usp_UpdateSalesHistory
(@id int, @date datetime)WITH EXECUTE AS OWNER, ENCRYPTIONAS EXTERNAL NAME
SalesHistory.UpdateBEGIN -- Code to retrieve data from SalesHistory and update
the desired table
END;
```

Answer: C
Section: (none)

Explanation/Reference:
The `EXECUTE AS` clause is used to specify the security context under which the stored should execute. You can specify the following values in the `EXECUTE AS` clause:
`SELF`: The stored procedure executes under the security context of the current user.
`OWNER`: The stored procedure executes under the security context of the user that owns the procedure.
`CALLER`: The stored procedure executes under the security context of the user calling the procedure. To execute the stored procedure successfully, the user calling the stored procedure would require permissions on the stored procedure and any underlying database objects referenced by the stored procedure.
`user_name`: The stored procedure executes under the security context of the specified user, regardless of which user called the stored procedure.

The `ENCRYPTION` clause encrypts the `CREATE PROCEDURE` statement used to create the stored procedure. This ensures that the statement is not stored in plain text that is readable by others. When a stored procedure's definition is encrypted, users cannot use the **sp_helptext** system stored procedure to view the stored procedure's definition. With the given `CREATE PROCEDURE` statement, if you attempted to use the **sp_helptext** system stored procedure to access the definition of the stored procedure, you would receive the following output:

```
The text for object 'dbo.usp_UpdateSalesHistory' is encrypted.
```

In addition, users cannot view the definition of the stored procedure by querying the **sys.sql_modules** catalog view or using Visual Designer in SQL Server Management Studio. In this scenario, if a user queried **sys. sql_modules** using the following `SELECT` statement, a definition value of `NULL` would be returned:

```
SELECT definition FROM sys.sql_modulesWHERE object_id = OBJECT_ID('dbo.
usp_UpdateSalesHistory');
```

You should not use the `CREATE PROCEDURE` statement that omits the `EXECUTE AS` clause. If you omit the `EXECUTE AS` clause when creating a stored procedure, then by default the stored procedure will execute under the security context of the caller. This statement also includes the `RECOMPILE` clause, which will cause the stored procedure to be recompiled each time it is called. You can use the `RECOMPILE` clause to force stored procedure compilation, but this was not required in this scenario.

You should not use the `CREATE PROCEDURE` statement that specifies a user name in the `EXECUTE AS` clause because this will cause the stored procedure to execute under the security context of **DevUser**.

You should not use the `CREATE PROCEDURE` statement that includes the `EXTERNAL NAME` clause because this statement will generate a syntax error. The `EXTERNAL NAME` clause is only valid when creating a CLR stored procedure. A CLR stored procedure is a stored procedure written in a .NET Framework language. To create a CLR stored procedure, you use the .NET language and define a static class method. Then, you compile the class, register the assembly in SQL Server with the `CREATE ASSEMBLY` statement, and use the `CREATE PROCEDURE` statement in SQL Server to reference the assembly. The `EXTERNAL NAME` clause must be specified in the `CREATE PROCEDURE` statement to reference the appropriate method.

QUESTION 115

You are a database developer on an instance of SQL Server 2008. You are creating a Service Broker application. You have created the **Message1** and **Message2** message types with the required validation. You create a contract using the following statement:

```
CREATE CONTRACT MyContract (Message1 SENT BY INITIATOR,Message2 SENT BY TARGET);
```

You want to create a queue to be used by a Service Broker service. The solution should meet the following requirements:

You want the queue to be able to receive messages.

You want all messages received on the queue to be immediately processed by the **ReceiveData** stored procedure.

You want the **ReceiveData** stored procedure to execute under the security context of the current user.

Which Transact-SQL statement should you use to create the queue?

A.
```
CREATE QUEUE MyQueue
WITH STATUS=OFF,ACTIVATION (STATUS=ON,PROCEDURE_NAME=ReceiveData,
MAX_QUEUE_READERS=3,EXECUTE AS SELF);
```

B.
```
CREATE QUEUE MyQueue
WITH STATUS=ON,ACTIVATION (STATUS=ON,PROCEDURE_NAME=ReceiveData,
MAX_QUEUE_READERS=3, EXECUTE AS OWNER);
```

C.
```
CREATE QUEUE MyQueue
WITH STATUS=ON,ACTIVATION (STATUS=OFF,PROCEDURE_NAME=ReceiveData,
MAX_QUEUE_READERS=3,EXECUTE AS SELF);
```

D.
```
CREATE QUEUE MyQueue
WITH STATUS=ON,ACTIVATION (STATUS=ON,PROCEDURE_NAME=ReceiveData,
MAX_QUEUE_READERS=3,EXECUTE AS SELF);
```

Answer: D
Section: (none)

Explanation/Reference:

You should use the following Transact-SQL statement to create the queue:

```
CREATE QUEUE MyQueueWITH STATUS=ON,ACTIVATION (STATUS=ON,
PROCEDURE_NAME=ReceiveData,MAX_QUEUE_READERS=3,EXECUTE AS SELF);
```

Before creating a service, you must execute the `CREATE QUEUE` statement to create a queue. A queue stores the incoming message for a service. Each service must be associated with a queue. Therefore, you must create a queue using the `CREATE QUEUE` statement before creating a service. The syntax of the `CREATE QUEUE` statement is as follows:

```
CREATE QUEUE queue_name [WITH [STATUS = {ON | OFF}],[RETENTION = {ON | OFF}],
[ACTIVATION ([STATUS = {ON | OFF},]PROCEDURE_NAME = procedure_name,
MAX_QUEUE_READERS = max_num_readers,EXECUTE AS {SELF | 'user_name' | OWNER})]][ON
{filegroup | [DEFAULT]}];
```

The clauses of the `CREATE QUEUE` statement are as follows:
`WITH STATUS`: Specifies the status of the queue that indicates whether the queue receives messages. With `STATUS=ON`, the queue receives the messages.
`RETENTION`: Specifies whether the sent and received messages will be retained in the queue until conversations have ended.
`ACTIVATION`: Specifies information about the stored procedure that is activated to process the message. The `ACTIVATION` clause can specify a `STATUS` option to indicate if the stored procedure immediately processes the message, and a `PROCEDURE_NAME` option with the name of the stored procedure that should process the message. The `ACTIVATION` clause can also specify the security context under which the stored procedure should execute using the `EXECUTE AS` clause, and the maximum number of instances of the stored procedure that can be concurrently used when processing messages.

In this scenario, you should create a queue and specify an `ACTIVATION` clause in the `CREATE QUEUE` statement. The `ACTIVATION` clause allows you to specify the stored procedure that will process incoming messages, and how it will behave. In the `ACTIVATION` clause, you should specify a `PROCEDURE_NAME` of `ReceiveData` to identify the procedure that will process incoming messages, and a `STATUS` of `ON` to indicate that the stored procedure should process the messages as soon as they are received. Finally, you should specify `EXECUTE AS SELF` to indicate that the stored procedure should execute under the security context of the current user.
You should not use the statement that specifies `WITH STATUS=OFF` when creating the queue. This setting indicates that the queue cannot receive messages.
You should not use the statement that specifies `STATUS=OFF` in the `ACTIVATION` clause of the `CREATE QUEUE` statement. With this setting, Service Broker will receive each message but will not process it immediately.
You should not use the statement that specifies `EXECUTE AS OWNER` in the `ACTIVATION` clause of the `CREATE QUEUE` statement. This would cause the stored procedure to execute under the security context of the queue owner, not of the current user as required in this scenario.

QUESTION 116
You are a database administrator on an instance of SQL Server 2008. You need to create an **Event** table that will store the following data:
A fixed-length, three-character alphanumeric code identifying the type of event
The event description, with a maximum length of 30 characters
The number of seats available for the event, with a maximum value of 50
A flag to indicate if the event is public or private
A event rating between 0 and 10 with two decimal places to the right of the decimal

Your table will only be used by English-speaking users, and you want to minimize the storage requirements for the table.

```
EventID int,Code varchar(3),Description varchar(30),SeatsAvail int,TypeFlag int,
Rating decimal(4,2));
```

Which `CREATE TABLE` statement should you use?

A.
```
CREATE TABLE Event (
EventID int,Code char(3),Description varchar(30),SeatsAvail smallint,TypeFlag
bit,Rating decimal(3,2));
```

B. `CREATE TABLE Event (`
`EventID int,Code char(3),Description varchar(30),SeatsAvail int,TypeFlag bit,`
`Rating decimal(4,2));`

C. `CREATE TABLE Event (`
`EventID int,Code char(3),Description varchar(30),SeatsAvail tinyint,TypeFlag`
`bit,Rating decimal(3,2));`

D. `CREATE TABLE Event (EventID int,Code char(3),Description varchar(30),SeatsAvail`
`tinyint,TypeFlag bit,Rating decimal(4,2));`

Answer: D
Section: (none)

Explanation/Reference:
You should use the following `CREATE TABLE` statement:

```
CREATE TABLE Event (EventID int,Code char(3),Description varchar(30),SeatsAvail
tinyint,TypeFlag bit,Rating decimal(4,2));
```

The **char** and **varchar** data types both allow character data up to 8,000 characters. The **char** data type is fixed-length, and the **varchar** data type is variable-length. Creating the **Code** column as `char(3)` will allow the **Code** column to store a fixed-length three-character alphanumeric code. Creating the **Description** column as `varchar(30)` allows for some storage savings for rows with descriptions shorter than 30 characters. You should use a **char** data type when values in a column will be a consistent length and a `varchar` data type when the length of values will vary.
For the **SeatsAvail** column, a **tinyint** data type would allow the required values but minimize storage. A **tinyint** data type can store an integer value between 0 and 255 with a single byte. The **bit** data type would be used to represent a Boolean value, such as whether the event is public or private. The **Rating** column is defined as `decimal(4,2)`. This allows the column to store a decimal value with a maximum of four total digits, with two digits to the right of the decimal place. This will allow you to store **Rating** values between 0.00 and 99.99.
You should not use the following `CREATE TABLE` statement:

```
CREATE TABLE Event (EventID int,Code varchar(3),Description varchar(30),
SeatsAvail int,TypeFlag int,Rating decimal(4,2));
```

A **varchar** data type stores variable-length data, rather than fixed-length data as required for the **Code** column in this scenario. The **SeatsAvail** column does not minimize storage because it is defined as an **int** data type. An **int** requires four bytes of storage. In this scenario, a **tinyint** would be sufficient. The **TypeFlag** does not minimize storage because it uses an **int** data type. Because only two possible values need to be stored, a **bit** could represent this with less storage.
You should not use the following `CREATE TABLE` statement:

```
CREATE TABLE Event (EventID int,Code char(3),Description varchar(30),SeatsAvail
smallint,TypeFlag bit,Rating decimal(3,2));
```

The **SeatsAvail** column does not minimize storage because it is defined as a **smallint** data type. A **smallint** data type requires two bytes of storage and can hold values between -32,768 and 32,767. A**tinyint** uses one byte and can store values between 0 and 255, which would have been sufficient in this scenario. The **Rating** column will not be able to store the required value because only three total digits are allowed. With this column definition, the **Rating** column could only hold values between 0.00 and 9.99.
You should not use the following `CREATE TABLE` statement:

```
CREATE TABLE Event (EventID int,Code char(3),Description varchar(30),SeatsAvail
tinyint,TypeFlag bit,Rating decimal(3,2));
```

With this statement, the **Rating** column will not be able to store the required value because only three total digits are allowed. With this column definition, the **Rating** column could only hold values between 0.00 and 9.99.

QUESTION 117
You are creating an application that will be used by several divisions of your company to record customer survey information. Divisions of the company are located in different time zones throughout the world. In the **SurveyDate** column of the **Survey** table, you want to store the date and time each survey was completed,

including time zone information.
You also want to record the amount of time each respondent took to complete the survey in the **ElapsedTime** column in hh:mm:ss format.

```
SurveyID int PRIMARY KEY,SurveyDate datetime,ElapsedTime smalldatetime);
```

Which `CREATE TABLE` statement should you use?

A. ```
   CREATE TABLE Survey (
   SurveyID int PRIMARY KEY,SurveyDate datetimeoffset,ElapsedTime time);
   ```
B. ```
   CREATE TABLE Survey (
   SurveyID int PRIMARY KEY,SurveyDate datetime2,ElapsedTime time);
   ```
C. ```
   CREATE TABLE Survey (
   SurveyID int PRIMARY KEY,SurveyDate datetimeoffset,ElapsedTime time(4));
   ```
D. ```
   CREATE TABLE Survey (
   SurveyID int PRIMARY KEY,SurveyDate datetime2,ElapsedTime time(4));
   ```

Answer: A
Section: (none)

Explanation/Reference:
SQL Server 2008 introduces four new date and time data types: **datetimeoffset**, **time**, **date**, and **datetime2**. The **datetimeoffset** data type includes time zone information in the form of a time zone offset. The time zone offset, which is in the format of +hh:mm or -hh:mm, ranges from
14:00 to +14:00. The time zone offset indicates the difference between Coordinated Universal Time (UTC) and the local time. For a positive time zone offset, the local time is calculated by adding the time zone offset to UTC. For a negative time zone offset, the local time is calculated by subtracting the time zone offset from UTC. For example, a time zone offset of -11:00 indicates that the date/time uses a local time zone that is eleven hours behind UTC. The **time** data type allows you to store only a time, including hours, minutes, seconds, and optionally nanoseconds. The **date** data type allows you to store only a date, and the **datetime2** data type is similar to the **datetime** data type but has greater range and precision.
In this scenario, you wanted to store the survey date to include the time zone information, so you should define the **SurveyDate** column as a **datetimeoffset** data type. The **datetimeoffset** data type is the only data type that is time-zone aware. You also wanted to define the **ElapsedTime** column to include only a time. To accomplish this, you would define the **ElapsedTime** column as a **time** data type. A **time** data type stores only a time in hh:mm:ss[n] format. A precision can be specified to store fractional seconds. In this scenario, no fractional seconds were required, so the default precision can be used.
You should not use the statement that defines the **SurveyDate** column as a **datetime** data type and the **ElapsedTime** column as a **smalldatetime** data type. The **datetime** data type is not time-zone aware as required in this scenario. Also, only the time portion is required for the **ElapsedTime** column in this scenario; therefore, a **time** data type should be used. The **smalldatetime** data type stores both the date and the time with the time not including any fractional seconds.
You should not use the statement that defines the **SurveyDate** column as a **datetime2** data type. The **datetime2** data type is not time-zone aware, as required in this scenario.
You should not use the statement that defines the **ElapsedTime** column using `time(4)`. In this scenario, you only needed to store the time in hh:mm:ss format without any fractional seconds. Therefore, specifying a precision is not required.

QUESTION 118
You are a database administrator on an instance of SQL Server. While creating a table, you decide to implement a sparse column.
Which Transact-SQL will execute successfully and create a table with a sparse column?

A.
```
CREATE TABLE Table1 (
ID int IDENTITY(1,1),
TranDate datetime,
ProductID int SPARSE NOT NULL,
Quantity smallint,
ProdType char(2),
ProdDetails varchar(max));
```
B.
```
CREATE TABLE Table1 (
ID int SPARSE NULL IDENTITY(1,1),
TranDate datetime,
ProductID int,
Quantity smallint,
ProdType char(2),
ProdDetails varchar(max));
```
C.
```
CREATE TABLE Table1 (
ID int IDENTITY(1,1),
TranDate datetime,
ProductID int NOT NULL,
Quantity smallint,
ProdType char(2),
ProdDetails varbinary(max) FILESTREAM SPARSE);
```
D.
```
CREATE TABLE Table1 (
ID int IDENTITY(1,1),
TranDate datetime,
ProductID int SPARSE NULL,
Quantity smallint,
ProdType char(2),
ProdDetails varchar(max));
```

Answer: D
Section: (none)

Explanation/Reference:
```
Pojawiła się odpowiedź A
```

QUESTION 119
You are a database developer. You plan to design a database solution by using SQL Server 2008. There are two schemas named Sales and Marketing. You are the owner of the Sales schema and the Marketing schema is owned by a user named MarketingUser. Users of the Marketing schema do not have permissions to access the Sales schema. You have permissions to create objects in all schemas in the database.

The Sales schema has a table named Customers.
You plan to create a stored procedure in the Marketing schema for the marketing team. The stored procedure will select data from the Customers table and will be owned by MarketingUser.

You need to ensure that the marketing team is able to execute the stored procedure.
What should you do?

A. Create the procedure by using the EXECUTE AS SELF option.
B. Create the procedure by using the EXECUTE AS CALLER option.
C. Create the procedure by using the EXECUTE AS OWNER option.
D. Create the procedure by using the EXECUTE AS USER=MarketingUser option.

Answer: A
Section: (none)

Explanation/Reference:

QUESTION 120
You are a database developer on an instance of SQL Server 2008 named **Srv1**. You need to define a stored procedure that will be used by one of
your applications. The stored procedure executes multiple DML statements to perform several table updates.
You want to ensure that SQL Server never caches query execution plans when executing this stored procedure.
Which action should you take?

A. Create the stored procedure with schema binding.
B. Call the **sp_recompile** system stored procedure.
C. Include the `WITH RECOMPILE` option when creating the stored procedure.
D. Use the `RECOMPILE` query hint within the stored procedure.

Answer: C
Section: (none)

Explanation/Reference:
You should include the `WITH RECOMPILE` option when creating the stored procedure. You can use the `WITH RECOMPILE` clause when creating a
stored procedure. This will force SQL Server to recompile the stored procedure each time it is called. SQL Server will not cache and reuse
execution plans. Using the `WITH RECOMPILE` clause to force compilation can improve performance for stored procedures that accept varying input
parameters that might result in different execution plans. However, you should note that using `WITH RECOMPILE` to always force stored procedure
recompilation might decrease performance.
You can use the `RECOMPILE` query hint to force recompilation
each time a single query is executed so that the cached plan for the query will not be used. However, in this scenario, the stored procedure contains
multiple queries, and you would have to specify the query hint for each query.
The **sp_recompile** system stored procedure will force recompilation of a stored
procedure, but only the next time the stored procedure is executed.
If you include the `WITH SCHEMABINDING` clause in a `CREATE PROCEDURE` statement, an error similar to the following occurs:
```
Msg 487, Level 16, State 1, Procedure MyProc, Line 3
An invalid option was specified for the statement "CREATE/ALTER PROCEDURE".
```
Schema binding is used when creating a view or function by including the `WITH SCHEMABINDING` clause in the `CREATE VIEW` or `CREATE FUNCTION` statement. Using schema binding ensures that no objects on which the view or function depends are dropped or modified in a way that
might affect the referencing object's definition. For example, including the `WITH SCHEMABINDING` clause in a `CREATE VIEW` statement ensures
that no base tables on which the view is based are dropped or modified in a way that might affect the view's definition. This ensures that users
cannot drop or modify base tables in such a way that the view becomes unusable. To drop base tables or make such modifications, you would first
need to drop the view, alter the view omitting `SCHEMABINDING`, or alter the view definition to remove any unwanted dependencies. Similarly, you
can include the `WITH SCHEMABINDING` clause when creating a function to ensure that objects referenced within the function are not changed in
such a way that would render the function unusable.

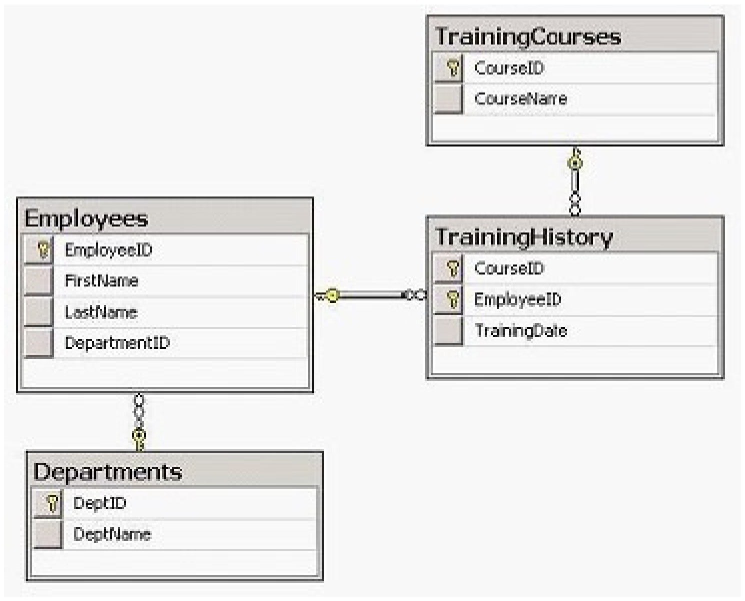
QUESTION 121
You are designing a SQL Server 2008 database that the human resources department will use to store information about employees and training courses each employee has completed.
You must design a database schema that will allow information about each employee's completed training to be recorded in the database, and will track completed training by department without introducing redundant data.
Your data model must also meet the following requirements:
Many training courses may be offered to employees.
Each employee may take one or more training courses.
Each training course may be taken by multiple employees.
Each employee may take a specific course only once.
Each employee is assigned to a single department.
Departments may have multiple employees.

Which actions should you take?

A. Create two tables, **Employees** and **TrainingCourses**, each containing a primary key.
Define a foreign key on **TrainingCourses** that references **Employees**
B. Create three tables, **Employees**, **Departments**, and **TrainingCourses**, each containing a primary key.
Define a foreign key on **Employees** that references **TrainingCourses**.
Define a foreign key on **Employees** that references **Departments**.
C. Create four tables, **Employees**, **Departments**, **TrainingCourses**, and **TrainingHistory**, each containing a primary key.
Define a foreign key on **Employees** that references **TrainingHistory**.
Define a foreign key on **TrainingCourses** that references **TrainingHistory**.
D. Create four tables, **Employees**, **Departments**, **TrainingCourses**, and **TrainingHistory**, each containing a primary key.
Define a foreign key on **Employees** that references **Departments**.
Define a foreign key on **TrainingHistory** that references **Employees**.
Define a foreign key on **TrainingHistory** that references **TrainingCourses**

Answer: D
Section: (none)

Explanation/Reference:
You can create the data model using the following Transact-SQL:
```
CREATE TABLE Departments(DeptID int PRIMARY KEY,DeptName varchar(25));
CREATE TABLE TrainingCourses(CourseID int PRIMARY KEY,CourseName varchar(30));
CREATE TABLE Employees (EmployeeID int PRIMARY KEY,FirstName varchar(25),LastName
varchar(30),DepartmentID int FOREIGN KEY REFERENCES Departments(DeptID));
CREATE TABLE TrainingHistory(CourseID int,EmployeeID int,TrainingDate datetime,
CONSTRAINT PK_THistoryPRIMARY KEY CLUSTERED (CourseID, EmployeeID),FOREIGN KEY
(CourseID) REFERENCES TrainingCourses(CourseID),FOREIGN KEY (EmployeeID)
REFERENCES Employees(EmployeeID));
```
The data model in this scenario will resemble the following:



A `PRIMARY KEY` constraint is used to create a primary key for a table that uniquely identifies each row in the table. A `FOREIGN KEY` constraint is used to establish a relationship between a primary key or a unique key in one table and another column in the same table or a different table.
Each row in the **Departments** table represents a single department identified by the **DeptID** primary key.
Each row in the **Employees** table represents a single employee identified by the **EmployeeID** primary key.
The `FOREIGN KEY` constraint in the **Employees** table that references the **Departments** table ensures that each employee is assigned to a single department, but allows departments to have multiple employees. Each row
in the **TrainingCourses** table represents a single course offered to employees, identified by the **CourseID** primary key.
To record the fact that an employee has completed a course, you must create an additional table, referred to as a junction table, to represent the many-to-many relationship between **Employees** and **TrainingCourses**.
The junction table contains a composite primary key consisting of the primary keys of the joined tables, and has `FOREIGN KEY` constraints on each of the primary key columns to reference the original tables. The junction table may also include any other applicable columns. Each row in the **TrainingHistory** table represents the fact that a specific employee has taken a specific training course. The composite primary key on the **CourseID** and **EmployeeID** columns in the **TrainingHistory** table ensures that an employee can take a specific course only once, and allows each training course to be taken by multiple employees. The foreign key that references the **EmployeeID** column in the **Employees** table ensures that only employees can take the offered training courses. The foreign key that references the **CourseID** column in the **TrainingCourses** table ensures that employees may only take courses that are offered.
All of the other options are incorrect because they will introduce redundant data into the data model or do not support the business requirements.
You should not create only two tables, **Employees** and **TrainingCourses**. If you only created two tables, redundant data would be introduced. For example, you might store department data in the **Employees** table. If so, because departments have multiple employees, the **DeptName** would be redundant for employees within a given department. In a normalized data model, each table contains data for a single entity, such as an employee, department, or course.
You should not create only three tables, **Employees**, **Departments**, and **TrainingCourses**. You must have a **TrainingHistory** table to represent the fact that a specific employee has taken a specific training course.

QUESTION 122
You are a database developer on an instance of SQL Server 2008. Your database contains the **Employee** table defined using the following Transact-SQL statement:
```
CREATE TABLE dbo.Employee(EmpID int PRIMARY KEY,LastName varchar(35) NOT NULL,
FirstName varchar(35) NOT NULL,HireDate datetime DEFAULT (GETDATE()),Status
varchar(8) CHECK (Status IN ('Active', 'Inactive')),SSN char(9) NOT NULL,Salary
money NOT NULL CHECK (Salary > 0),Commission money DEFAULT 0);
```
You want to create a view that meets the following requirements:
Users must not be able to query data for inactive employees.
Users must not be able to query the **SSN**, **Salary**, or **Commission** column for any employees.

In addition, developers must not be able to modify the structure of the **Employee** table such that the view becomes unusable.
Which statement should you use to create the view?

A.
```
CREATE VIEW EmpView
WITH SCHEMABINDING AS SELECT * FROM dbo.Employee WHERE Status <> 'Inactive';
```
B.
```
CREATE VIEW EmpView
AS SELECT EmpID, LastName, FirstName, HireDate, Status
FROM dbo.Employee WHERE Status <> 'Inactive';
```
C.
```
CREATE VIEW EmpView
WITH SCHEMABINDING AS SELECT EmpID, LastName, FirstName, HireDate, Status
FROM dbo.Employee;
```
D.
```
CREATE VIEW EmpView
WITH SCHEMABINDING AS SELECT EmpID, LastName, FirstName, HireDate, Status
FROM dbo.Employee WHERE Status <> 'Inactive';
```

Answer: D
Section: (none)

Explanation/Reference:
```
CREATE VIEW EmpViewWITH SCHEMABINDING AS SELECT EmpID, LastName, FirstName,
HireDate, Status FROM dbo.EmployeeWHERE Status <> 'Inactive';
```
A view is a logical table based on a `SELECT` statement that references one or more underlying tables or views. When you create a view, the `SELECT` statement defining the view is stored in the database, rather than the actual data. When a user references the view in a query, the values are retrieved from the underlying tables or views using the view's `SELECT` statement. This ensures that the data that is visible to users through the
view is always the most recent data.
Views are often used to restrict access to specific columns and rows of data in a table. When defining a view, you can prevent users from accessing a column by omitting the column from the `SELECT` list. You can prevent users from accessing specific rows by including a `WHERE` clause that filters the rows that are visible through the view. For example, in this scenario, users cannot query the **SSN**, **Salary**, or **Commission** columns because these columns are not included in the `SELECT` list. In addition, users cannot query inactive employees because the `WHERE` clause filters out employees that are inactive.
In this scenario, you also wanted to prevent developers from making changes to the table that might make the view unusable. You should include the `WITH SCHEMABINDING` clause to accomplish this. The `WITH SCHEMABINDING` clause ensures that base tables of a view cannot be dropped or modified in a way that affects the view's definition. This prevents users from dropping or modifying base tables in such a way that the view becomes unusable. To drop base tables or make such modifications, you would need to first drop the view, alter the view omitting `SCHEMABINDING`, or alter the view to remove any unwanted dependencies. To use schema binding for views, you must include a `SELECT` column list and all objects must be referenced using their two-part name, which includes the schema name and object name.
You should not use the statement that includes the `WITH SCHEMABINDING` clause but uses an asterisk (`*`) in the `SELECT` list because when schema binding a view, the `SELECT` list must contain a list of columns. This statement will generate the following error:
```
Msg 1054, Level 15, State 6, Procedure EmpView, Line 5Syntax '*' is not allowed
in schema-bound objects.
```
You should not use the statement that omits the `WITH SCHEMABINDING` clause because this will not meet the requirement of preventing developers from making database changes that might affect the view's definition.
You should not use the statement that omits the `WHERE` clause because this will not meet the requirement of only allowing users to query active employees.

QUESTION 123

a database developer on an instance of SQL Server 2008. Your **Prod** database contains a **Meeting** table defined using the following statement:

```
CREATE TABLE Meeting (ID int IDENTITY(1,1) PRIMARY KEY,
Description varchar(30) NOT NULL,MaxAttendance smallint DEFAULT 0,Type bit NULL,
Priority tinyint CHECK (Priority BETWEEN 0 and 10),Cost money NULL);
```

Assuming appropriate permissions, which statement will successfully create a view?

A.
```
CREATE VIEW RegularMeeting
WITH SCHEMABINDING AS SELECT * FROM MeetingWHERE Priority = 5WITH CHECK OPTION;
```

B.
```
CREATE VIEW SpecialMeeting
AS SELECT * FROM MeetingWHERE Priority = 1WITH CHECK OPTION;
```

C.
```
CREATE VIEW WeeklyMeeting
WITH SCHEMABINDING, ENCRYPTIONAS SELECT ID, DescriptionFROM MeetingWHERE
Priority = 7WITH CHECK OPTION;
```

D.
```
CREATE VIEW StatusMeeting
WITH SCHEMABINDING AS SELECT m.* FROM dbo.Meeting mWHERE Priority = 4;
```

Answer: B
Section: (none)

Explanation/Reference:
Assuming appropriate permissions, the following statement will successfully create a view:

```
CREATE VIEW SpecialMeetingAS SELECT * FROM MeetingWHERE Priority = 1WITH CHECK
OPTION;
```

This statement will create a view named **SpecialMeeting**, through which all columns in the **Meeting** table for rows with a **Priority** value of 1 are visible. The `WITH CHECK OPTION` clause is used to ensure that no data modifications can be made through a view that would cause the underlying data to violate the view's definition. Because you included the `WITH CHECK OPTION` clause, an `UPDATE` or `INSERT` statement that updates data using the view cannot modify the data in a way that would cause the data to violate the view's definition. For example, if you attempted to update a row using the view that sets the **Priority** value to a value other than 1, an error message similar to the following would be displayed:

```
Msg 550, Level 16, State 1, Line 2The attempted insert or update failed because
the target view either specifies WITH CHECK OPTION or spans aview that specifies
WITH CHECK OPTION and one or more rows resulting from the operation did not
qualify under the CHECK OPTION constraint.
```

The `CREATE VIEW` statements for the **RegularMeeting** and **StatusMeeting** views include the `WITH SCHEMABINDING` clause and an asterisk (*) for the `SELECT` list. Each of these statements will generate an error similar to the following because you must explicitly list the columns in the `SELECT` list when using schema binding:

```
Msg 1054, Level 15, State 6, Procedure SpecialMeeting, Line 5Syntax '*' is not
allowed in schema-bound objects.
```

The `WITH SCHEMABINDING` clause ensures that base tables of a view cannot be dropped or modified in a way that affects the view's definition.
This prevents users from dropping or modifying base tables in a way that makes the view unusable. To drop base tables or make such modifications, you would need to first drop the view, alter the view omitting `SCHEMABINDING`, or alter the view to remove any unwanted dependencies.
The `CREATE VIEW` statement for the **WeeklyMeeting** view will generate the following error:

```
Msg 4512, Level 16, State 3, Procedure WeeklyMeeting, Line 5Cannot schema bind
view 'WeeklyMeeting' because name 'Meeting' is invalid for schema binding. Names
must bein two-part format and an object cannot reference itself.
```

The statement fails because only the object name was specified in the `SELECT` statement. When schema binding a view you must specify the two-part object name, including the schema name and object name. The `WITH ENCRYPTION` clause is used to encrypt the `CREATE VIEW` statement that is stored in the **sys.**

syscomments table. This ensures that the statement is not stored in plain text that is readable by others.
You should also note that when creating a view, with or without schema binding, the underlying `CHECK` constraints on base tables are ignored. For example, in this scenario, the following statement would successfully create a view with schema binding and encryption, but no rows would be visible through the view because of the table's `CHECK` constraint:

```
CREATE VIEW ManagerMeetingWITH SCHEMABINDING, ENCRYPTIONAS SELECT ID,
DescriptionFROM dbo.MeetingWHERE Priority = 12WITH CHECK OPTION;
```

QUESTION 124
You are a database developer. You plan to design a database solution by using SQL Server 2008. You have a database that contains a table and a table-valued function. The table-valued function accepts the primary key from the table as a parameter.
You plan to write a query that joins the table to the results of the table-valued function. You need to ensure that only rows from the table that produce a result set from the table-valued function are returned.
Which join predicate should you use?

A. CROSS APPLY
B. OUTER APPLY
C. INNER JOIN
D. LEFT OUTER JOIN

Answer: A
Section: (none)

Explanation/Reference:

QUESTION 125
You are a database developer for VirtuArt Corporation on an instance of SQL Server 2008. You are creating an application for the order processing department. The application will send e-mail notifications to sales representatives when specific orders for priority customers are shipped.
When an order from a priority customer that is over $1000.00 is shipped, the application must e-mail the customer's assigned sales representative and include details of the order as an e-mail attachment, and also meet the following requirements:
E-mail should be sent using Simple Mail Transfer Protocol (SMTP).
Impact on the SQL server should be minimized.

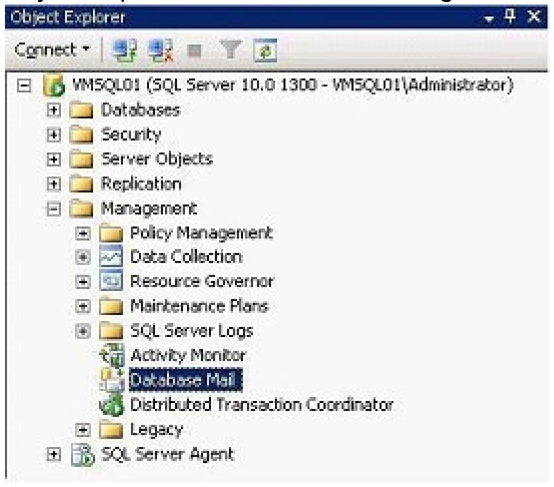
Which SQL Server component should you use?

A. Database Mail
B. SQL Server Mail
C. Distributed Transaction Coordinator (DTC)
D. SQL Server alerts

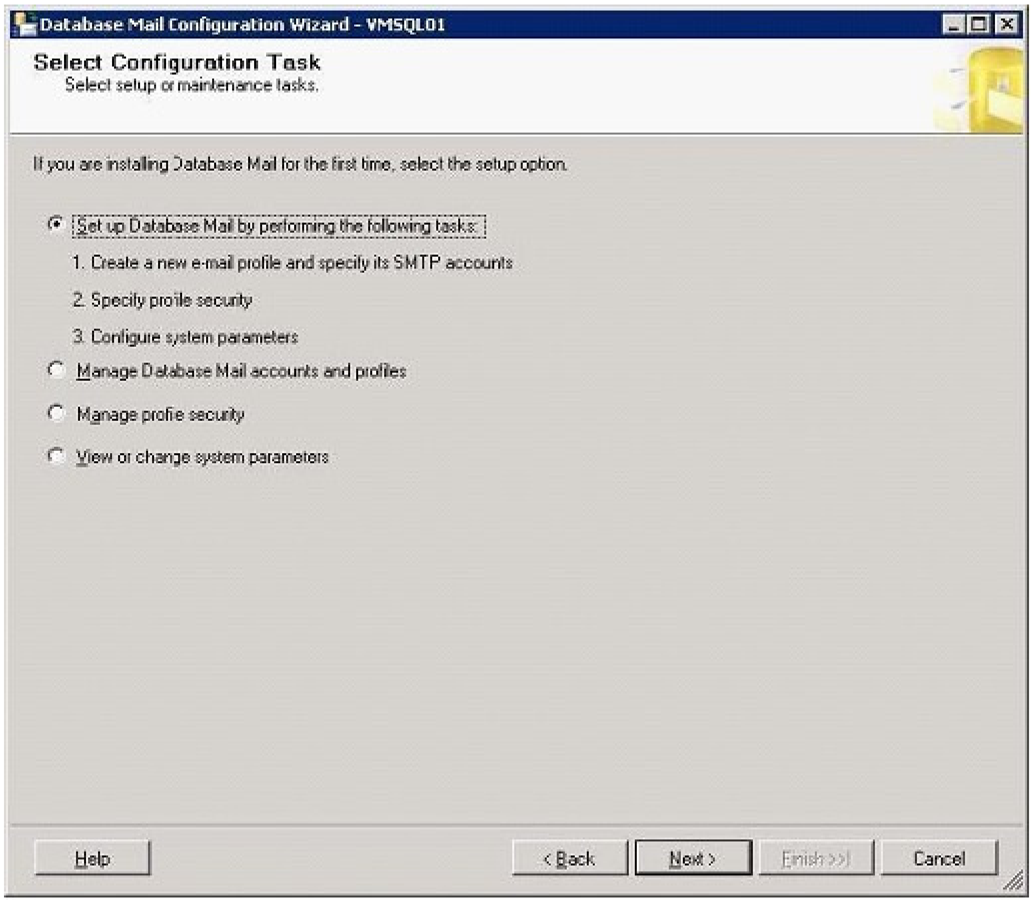
Answer: A
Section: (none)

Explanation/Reference:
Database Mail sends e-mail messages using SMTP. To use Database Mail, you do not have to install an Extended MAPI client on the SQL server. Database Mail is an external process that runs outside of the Database Engine. Therefore, it has minimal impact on the SQL server. Database Mail offers many advantages over using SQL Server Mail. Database Mail can use multiple SMTP accounts and profiles, limit the size and types of attachments, and log all mail events that occur.

In this scenario, you could include code in the application to send an e-mail to the assigned sales representative with an attachment containing the order details when an order over $1,000 ships to a priority customer. Mail messages sent by Database Mail use Service Broker to queue and deliver e-mail messages. Therefore, Service Broker must be active in the **msdb** database to successfully send e-mail messages using Database Mail.
Database Mail is disabled by default. You can enable Database Mail using the Database Mail Configuration Wizard in SQL Server Management Studio, the **sp_configure** system stored procedure, or the Surface Area Configuration Utility.
To launch the Database Mail Configuration Wizard, you should first expand the **Management** node in the Object Explorer in SQL Server Management Studio as shown:



After expanding the **Management** node, you should right-click **Database Mail** and select **Configure Database Mail**:
This will launch the Database Mail Configuration Wizard. At the first screen, click **Next** to proceed past the welcome information. At the second screen, select the **Setup Database Mail** option as shown and click the **Next** button.


Database Mail Configuration Wizard - VMSQL01

Select Configuration Task

Select setup or maintenance tasks.

If you are installing Database Mail for the first time, select the setup option.

(●) Set up Database Mail by performing the following tasks:

1. Create a new e-mail profile and specify its SMTP accounts
2. Specify profile security
3. Configure system parameters

( ) Manage Database Mail accounts and profiles

( ) Manage profile security

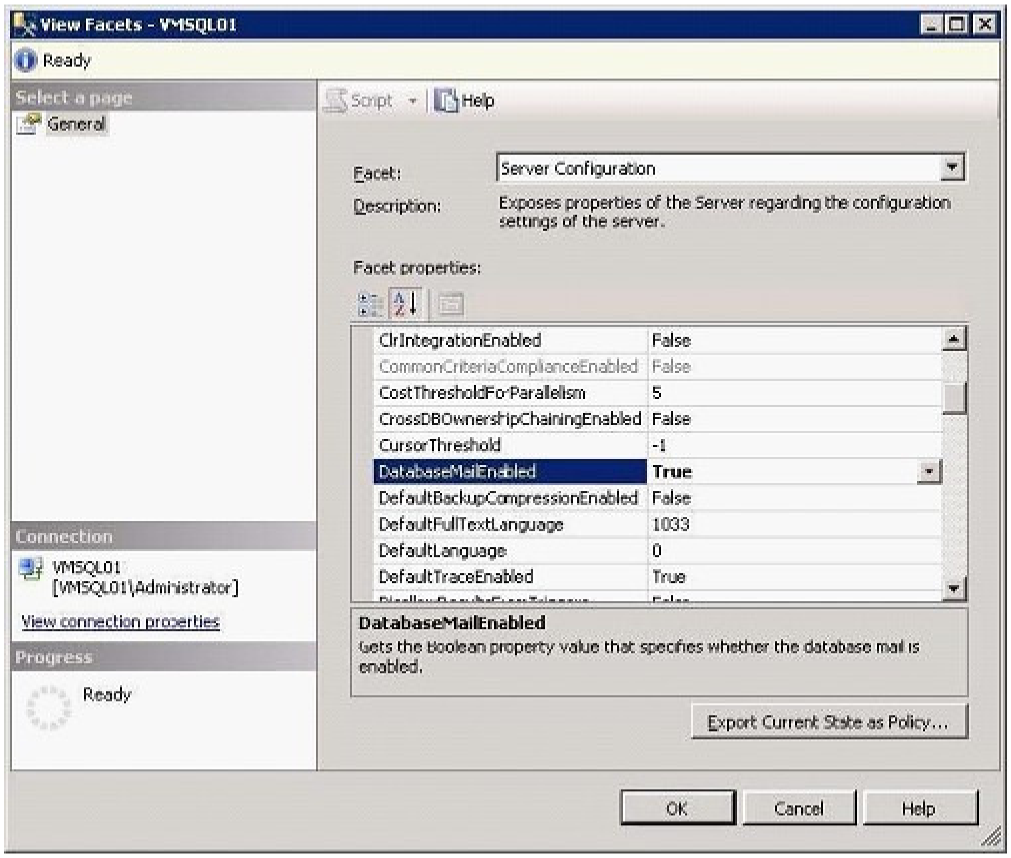
( ) View or change system parameters

The Database Mail Configuration Wizard will walk you through the initial steps of configuring database mail and will enable Database Mail.

To enable Database Mail using the **sp_configure** system stored procedure, you can enable the **Database Mail XPs** option as follows:

```
sp_configure 'show advanced options', 1;GO RECONFIGURE;GO
sp_configure 'Database Mail XPs', 1;GO RECONFIGURE GO
```

To enable Database Mail using the Surface Area Configuration Utility, you should right-click the server in Object Explorer and select the **Facets** option. In the **View Facets** window, you should select **Server Configuration** for the **Facet** option and set the **DatabaseMailEnabled** property to **True** as shown:



VMSQL01
[VMSQL01\Administrator]

Facet: Server Configuration

Description: Exposes properties of the Server regarding the configuration settings of the server.

Facet properties:

ClrIntegrationEnabled	False
CommonCriteriaComplianceEnabled	False
CostThresholdForParallelism	5
CrossDBOwnershipChainingEnabled	False
CursorThreshold	-1
DatabaseMailEnabled	True
DefaultBackupCompressionEnabled	False
DefaultFullTextLanguage	1033
DefaultLanguage	0
DefaultTraceEnabled	True

DatabaseMailEnabled
Gets the Boolean property value that specifies whether the database mail is enabled.

After enabling Database Mail, you can use Database Mail system stored procedures to manage mail accounts and profiles or send e-mail messages. For example, in this scenario, the following code might be used to send an e-mail to a sales representative using Database Mail:

```
EXEC msdb.dbo.sp_send_dbmail@recipients=N'asmith@virtuart.com',@body='An order >
$1000 for a priority customer has shipped.', @subject ='Priority Customer Order',
@profile_name ='VirtuArtMailProfile',@query ='SELECT * FROM Sales.
SalesOrderHeader h INNER JOIN Sales.SalesOrderDetail d ON h.SalesOrderID = d.
SalesOrderID WHERE TotalDue > 1000 AND CustType = 'Priority' AND SalesOrderID =
43652', @attach_query_result_as_file = 1,@query_attachment_filename
='PriorityOrderDetails.txt'
```

The **sp_send_dbmail** system stored procedure is used to send an e-mail message using Database Mail. This procedure accepts parameters defining the details of the e-mail message, such as the recipients, message subject, message body, and importance, and sends the e-mail message.

You should not use SQL Server Mail. SQL Server Mail requires an Extended MAPI client be installed on the server, and runs as a server process. This would not allow you to send messages using SMTP and would have more impact on the server than using Database Mail.

You should not use Distributed Transaction Coordinator (DTC). Microsoft Distributed Transaction Coordinator (MS DTC) is used to provide for transactional processing across multiple SQL Server instances.

You should not use SQL Server alerts. SQL Server alerts provide automatic notification when specific errors or events occur on the SQL server. When a server error or event occurs, it is recorded in the Windows application log. SQL Server Agent reviews the application log entries and fires alerts defined for the recorded events.

QUESTION 126

You are designing a data storage solution for a transactional application.You need to ensure that each row in a table records the date and the time that the row was written. The time must be as precise as possible.

Which data type should you use?

A. datetime

B. datetime2
C. smalldatetime
D. timestamp

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 127
Database used in three different timezones. What to do so that value can be read everywhere?

A. datatype datetimeoffset
B. datatype datetime2
C. function sysgetdatetimeoffset
D. function ?

Answer: A
Section: (none)

Explanation/Reference:

QUESTION 128
Your company requires a standard code snippet to be inserted at the beginning of all stored procedures meeting the following requirements:
- Stored procedures should be able to call other stored procedures
- All stored procedures must incorporate transactions

Here is the core part of the snippet

```
char(36) @TransName = newID()
If <insert answer here>
Save Transaction @TranName
Begin Transaction
```

A. @@DBTS = 0
B. @@FETCH_STATUS <= -1
C. @@NESTLEVEL = 0
D. @@TRANCOUNT > 1

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 129
You are a database administrator on an instance of SQL Server 2008. You are creating the **TrxHistory** table in the **Sales** database. You want to create a surrogate key for the table that will store a globally-unique identifier (GUID).

Users at your branch offices will insert many rows into the table, and users in the accounting department at your corporate office will frequently run queries against the table. You want to minimize the size and fragmentation for the clustered primary index on the **TrxHistory** table.
Which action should you take?

A. Define the surrogate key as an **int** and include the `IDENTITY` property.
B. Define the surrogate key as a **uniqueidentifier** and use the `NEWID` function in the column's `DEFAULT` definition.
C. Define the surrogate key as a **uniqueidentifier** and use the `NEWSEQUENTIALID` function in the column's `DEFAULT` definition.
D. Define the surrogate key column as a **uniqueidentifier** and include the `ROWGUIDCOL` property.

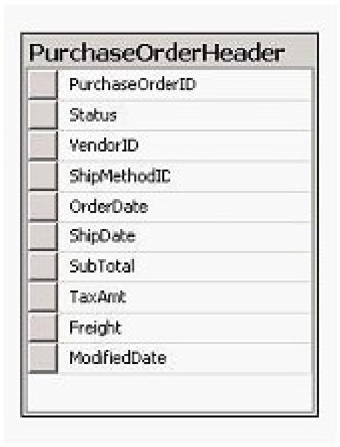
Answer: C
Section: (none)

Explanation/Reference:
A **uniqueidentifier** data type is used to store globally-unique identifiers (GUIDs). A GUID is a unique 16-byte integer that is unique across servers and networks. For **uniqueidentifier** columns, you can use the `NEWSEQUENTIALID` function in the column's `DEFAULT` definition to generate a GUID each time a row is inserted into the table. For example, you could use the following statement to create the **TrxHistory** table:

```
CREATE TABLE TrxHistory (ID uniqueidentifier PRIMARY KEY DEFAULT NEWSEQUENTIALID
(),Date datetimeoffset,Description nvarchar(50));
```

You can also use the `NEWID` function to generate a GUID. However, the `NEWSEQUENTIALID` function will generate sequential GUID values and minimize fragmentation of the index. You should note that the `NEWID` function can be also be used in the `VALUES` clause of an `INSERT` statement, but the `NEWSEQUENTIALID` function can only be used in a `DEFAULT` definition.
You should not define the surrogate key as an **int** and include the `IDENTITY` property. In this scenario, you wanted the column to store a globally-unique identifier. This cannot be accomplished using an **int** data type. You should use a **uniqueidentifier** data type to store a GUID.
You should not define the surrogate key as a **uniqueidentifier** and use the `NEWID` function in the column's `DEFAULT` definition. In this scenario, you wanted to minimize index size and fragmentation. Using the `NEWSEQUENTIALID` function to generate the GUIDs would generate them sequentially, instead of randomly, and minimize fragmentation on the index.
You should not define the surrogate key column as a **uniqueidentifier** and include the `ROWGUIDCOL` property. The `ROWGUIDCOL` property can be specified for a single **uniqueidentifier** column in a table to identify the column as a row GUID column. This allows you to use `$ROWGUID` to reference the column, but does not ensure uniqueness, nor affect how the GUID values are generated, and would not minimize index fragmentation in this scenario.

QUESTION 130
Your **Prod** database resides on a SQL Server 2008 instance. You have a **PurchaseOrderHeader** table defined as follows in the **Prod** database:



You open a session and execute the following Transact-SQL:

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION SELECT * FROM PurchaseOrderHeader;
```

You open another session.
Which statement will successfully execute in the second session with no blocking?

A. `DELETE FROM PurchaseOrderHeader`
   `WHERE PurchaseOrderID = 1;`

B. `UPDATE PurchaseOrderHeader`
   `SET ShipMethodID = 2`
   `WHERE PurchaseOrderID = 1;`

C. `ALTER TABLE PurchaseOrderHeader`
   `ADD ProjectID int;`

D. `INSERT INTO PurchaseOrderHeader(Status, VendorID, ShipMethodID,OrderDate,`
   `ShipDate, SubTotal, TaxAmt,`
   `Freight, ModifiedDate)VALUES(0,0,0,'02-13-2009',NULL, 10, 10, 10, GETDATE());`

Answer: D
Section: (none)

Explanation/Reference:
By default, SQL Server automatically handles locking and attempts to use the least restrictive locking applicable for the operation being performed. When a database operation is performed, SQL Server determines the resource that needs to be locked, and to what extent it should be locked. Based on what action is being performed, SQL Server can lock the row, key, page, extent, table, or the entire database. Each lock has a lock mode identifying how the lock will actually behave. These lock modes include Shared(S), Exclusive (X), Update (U), Intent, and Schema.
In this scenario, the transaction you started in the first session has a transaction isolation level of `REPEATABLE READ`. This causes SQL Server to acquire shared locks and hold them until the transaction ends. The `INSERT` statement you issued in the second session requires a shared lock. Therefore, it executes with no blocking.
All of the other options are incorrect because all of these statements will attempt to acquire an exclusive lock and will hang.

QUESTION 131
You are a database developer on an instance of SQL Server 2008. You are creating a view that will allow users to query information for customer orders. Some users have full database access. You want to prevent these users from viewing the `SELECT` statement that defines the view. Which action should you take?

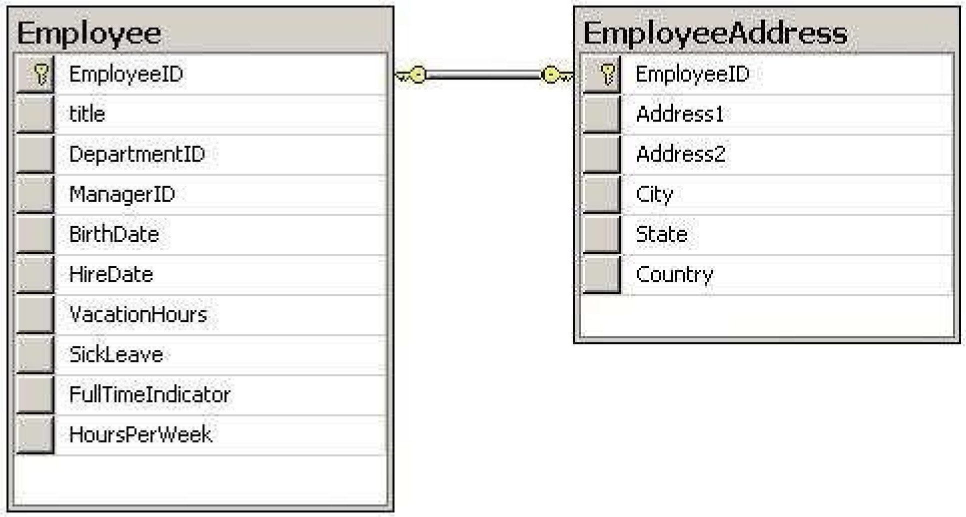
A. Delete the view from **sys.objects**.

B. Alter the view to remove schema binding.
C. Drop the view and re-create it including `WITH ENCRYPTION`.
D. Implement Transparent Data Encryption (TDE) for the view.

Answer: C
Section: (none)

Explanation/Reference:
The `WITH ENCRYPTION` clause is used to encrypt the `CREATE VIEW` statement that is stored in the **sys. syscomments** table. This ensures that the statement is not stored in plain text that is readable by others. When a view is encrypted, the view's definition cannot be accessed using the **sp_helptext** system stored procedure, directly queried from the **sys.sql_modules** catalog view, or accessed from the Visual Designer in SQL Server Management Studio.
You should not delete the view from **sys.objects**. You should never directly alter data in system objects, but rather allow SQL Server to manage them.
You should not alter the view to remove schema binding. Schema binding of a view ensures that the base tables used by the view remain usable; it does not control whether the statement that defines the view is encrypted or stored as plain text. The `WITH SCHEMABINDING` clause of the `CREATE VIEW` statement ensures that base tables of a view cannot be dropped or modified in a way that affects the view's definition. This prevents users from dropping or modifying base tables in such a way that the view becomes unusable. To drop base tables or make such modifications, you would need to first drop the view, alter the view omitting `SCHEMABINDING`, or alter the view to remove any unwanted dependencies.
You should not implement Transparent Data Encryption (TDE) for the view. TDE is a special type of full database encryption that uses a symmetric key to encrypt the entire database. It is not used to encrypt the statements that define views, functions, stored procedures, or triggers.

QUESTION 132
You are a database developer. You develop a database application for a SQL Server 2008 instance. The instance hosts a third-party database. You are not allowed to modify the database schema. The database contains two tables that are as shown in the following diagram.



You plan to extract address information about full-time employees based on the FullTimeIndicator flag. You need to design a data access layer to simplify the extraction process. What should you do?

A. Design an Entity Data Model that contains the EMPLOYEES and ADDRESS entities.
B. Create a view on the database to include full-time employees and their address details.
C. Re-design the underlying database model to include employee and address information in one table.

D. Design a conceptual Entity Data Model that contains an entity named EMPLOYEE_ADDRESS. Ensure that this entity contains information about employees and their addresses.

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 133
You are a database developer. You provide solutions by using SQL Server 2008 in an enterprise environment.
Your online transaction processing (OLTP) database contains a tabled named SalesOrders. Your data warehouse contains a table named factBuyingHabits. The factBuyingHabits table has no indexes.
You need to synchronize data between the two tables on a weekly basis. The synchronization process has the following requirements:
·
New records in the SalesOrders table are inserted in the factBuyingHabits table.
·
When a record is modified in the SalesOrders table, the modification is updated in the factBuyingHabits table.
·
Records that are deleted from the SalesOrders table are also deleted from the factBuyingHabits table.
You need to design an appropriate synchronization solution. You want to achieve this goal by using minimum amount of coding and administrative efforts.
What should you do?

A. Design an SSIS package each for the INSERT, UPDATE, and DELETE operations. Schedule a job to

run this package.
B. Design a single SSIS package that uses the Slowly Changing Dimension task. Schedule a job to run this package.
C. Write one stored procedure that contains a MERGE statement to perform the INSERT, UPDATE, and DELETE operations. Schedule a job to run the stored procedure.
D. Write three stored procedures each for the INSERT, UPDATE, and DELETE operations. Schedule a job to run the stored procedures in a sequential manner.

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 134
You are a database developer on an instance of SQL Server 2008. You have a **Product** table that contains a column-level `CHECK` constraint named **qty_check**, which verifies that the **QuantityOnHand** value is greater than or equal to zero.
A stored procedure runs each night to gather information from warehouse locations and update the **QuantityOnHand** column values. You have some reports that use historical data and include information for items that have been discontinued. For these discontinued items, you want to insert rows into the **Product** table that have a negative **QuantityOnHand**.
You want the stored procedure that runs each night to be unaffected. Which action should you take?

A. Disable the constraint and re-enable it after inserting the rows.
B. Drop the constraint, insert the rows, and re-create the constraint.

C. Drop the constraint and define the validation for the constraint in a DML trigger.
D. Re-create the constraint as a table-level constraint.

Answer: A
Section: (none)

Explanation/Reference:
You should disable the constraint and re-enable it after inserting the rows containing negative **QuantityOnHand** values. You can temporarily disable a `FOREIGN KEY` or `CHECK` constraint by including the `NOCHECK CONSTRAINT` clause in an `ALTER TABLE` statement. When a constraint is disabled, the constraint condition is not checked when data is inserted or updated. For example, in this scenario, you could use the following statement to disable the **qty_check** constraint:

```
ALTER TABLE Product NOCHECK CONSTRAINT qty_check;
```

After disabling the constraint, you could insert rows into the table that violate the constraint because the constraint will not be checked. Finally, you could use the following statement to re-enable the constraint:

```
ALTER TABLE Product CHECK CONSTRAINT qty_check;
```

You should not re-create the constraint as a table-level constraint. Whether a constraint is defined at the table level or the column level does not affect how validation is performed. All constraints defined at either level are enforced if they are enabled.
You should not drop the constraint and define the validation for the constraint in a DML trigger. Although you can perform complex validation within a DML trigger, this requires more development effort. In addition, the validation would still occur that would prevent you from inserting the rows. In this scenario, you should temporarily disable the `CHECK` constraint, insert the row, and then re-enable the `CHECK` constraint.
You should not drop the constraint, insert the rows, and re-create the constraint because this would require more effort. Instead, you should temporarily disable the constraint.

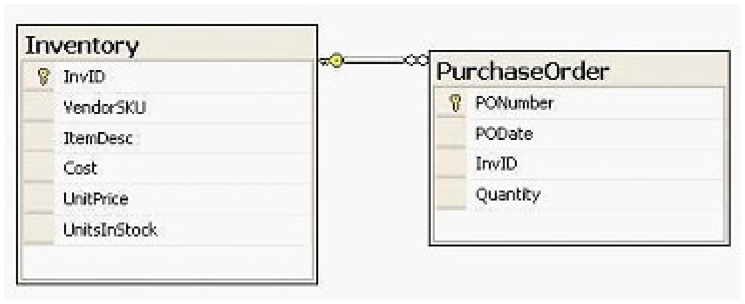
QUESTION 135
You are a database developer on an instance of SQL Server 2008. Your database contains a table named **ProductDetails** that has a `PRIMARY KEY` constraint defined on the **ProdID** column. Several other nonclustered indexes have been created on the table.
For a particular query, you discover that full table scans are being performed because the optimizer does not choose the best index. You want to force the optimizer to use the clustered index.
Which action should you take?

A. Drop all existing nonclustered indexes.
B. Use the `INDEX(0)` table hint.
C. Use the `FORCESEEK` table hint.
D. Use the `FORCE ORDER` query hint.

Answer: B
Section: (none)

Explanation/Reference:
If a clustered index exists on the table, the `INDEX(0)` hint forces a clustered index scan on the table. If no clustered index exists, the hint forces a full table scan. You can also use the `INDEX` table hint and specify a specific index that the query optimizer should use, or you can use the `INDEX(1)` to force an index scan or seek on the clustered index.
You should not use the `FORCE ORDER` query hint. The `FORCE ORDER` query hint controls how the query's join order is handled when a query is being optimized. Specifying `FORCE ORDER` indicates that query optimization will not affect the join order specified in the query.
You should not drop all existing nonclustered indexes because this might adversely affect other queries.
You should not use the `FORCESEEK` table hint. The `FORCESEEK` table hint can be specified to force the query optimizer to use only an index seek when accessing the data. The query optimizer will consider both clustered and nonclustered indexes, but will not use a specific index.

QUESTION 136
You are a database developer on an instance of SQL Server 2008. Your company is using a database to record its purchasing transactions. You have an **Inventory** table and a **PurchaseOrder** table defined as follows:



Currently, only one inventory item may be purchased on a single purchase order. You want to modify the data model so that each purchase order may contain multiple line items. You want to minimize data redundancy. Which action(s) should you take?

A. Drop the existing `FOREIGN KEY` constraint.
Create a **POLines** table that includes the **InvID** and **Quantity** columns.
Create a `FOREIGN KEY` constraint on the **InvID** column of the **POLines** table that references the **InvID** column of the **Inventory** table.

B. Create a `FOREIGN KEY` constraint from each **InvID** column that references the **InvID** column in the **Inventory** table.

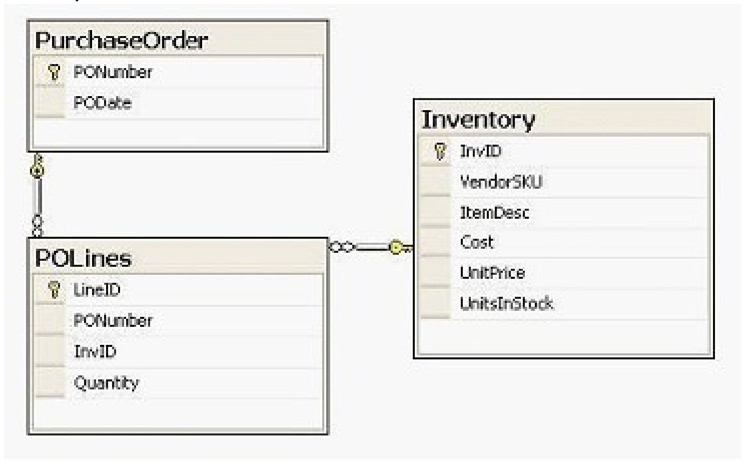
C. Drop the existing `FOREIGN KEY` constraint.
Remove the **InvID** and **Quantity** columns from the **PurchaseOrder** table.
Create a **POLines** table that includes the **PONumber**, **InvID**, and **Quantity** columns.
Create a `FOREIGN KEY` constraint on the **InvID** column of the **POLines** table that references the **InvID** column in the **Inventory** table.
Create a `FOREIGN KEY` constraint on the **PONumber** column of the **POLines** table that references the **PONumber** column of the **PurchaseOrder** table.

D. Denormalize the data model by combining the **Inventory** and **PurchaseOrder** tables into a single table.
Add multiple columns for **InvID** and **Quantity** values to the **PurchaseOrder** table.

Answer: C
Section: (none)

Explanation/Reference:
To accomplish the desired objectives, you should perform the following actions:
Drop the existing `FOREIGN KEY` constraint.
Remove the **InvID** and **Quantity** columns from the **PurchaseOrder** table.
Create a **POLines** table that includes the **PONumber**, **InvID**, and **Quantity** columns.
Create a `FOREIGN KEY` constraint on the **InvID** column of the **POLines** table that references the **InvID** column in the **Inventory** table.
Create a `FOREIGN KEY` constraint on the **PONumber** column of the **POLines** table that references the **PONumber** column of the **PurchaseOrder** table.

To record multiple line items on a single purchase order, you should create a separate table to store the lines of each purchase order. Each row in the **PurchaseOrder** table will represent a single purchase, and each row of the new **POLines** table will represent the purchase of a specific inventory item on a purchase order. After dropping the existing `FOREIGN KEY` constraint, you should move the **InvID** and **Quantity** columns from the **PurchaseOrder** table to the **POLines** table. Next, you should create a `FOREIGN KEY` constraint on the **InvID** column of the **POLines** table that references the **InvID** column in the **Inventory** table. This will ensure that purchase order lines can only be added for items that exist in the **Inventory** table. Finally, you should create a `FOREIGN KEY` constraint on the **PONumber** column of the **POLines** table to associate a purchase order with its respective line items. In this scenario, the revised data model would resemble the following:



You should not drop the existing `FOREIGN KEY` constraint, create a **POLines** table including the **InvID** and **Quantity** columns, and create a `FOREIGN KEY` constraint on the **InvID** column that references **InvID** column of the **Inventory** table. This would only accomplish a portion of the desired objectives. The resulting data model would not include a relationship to associate each purchase order with its respective line items. To do so, you would also have to include the **PONumber** column in the **POLines** table and create a relationship between the **PurchaseOrder** and **POLines** tables.

You should not denormalize the data model by combining the **Inventory** and **PurchaseOrder** tables into a single table because this data model would store redundant data. If you combined the **Inventory** and **PurchaseOrder** tables, the same inventory item information would have to be stored in multiple rows, once for each purchase order line containing that item. Multiple rows in the combined table would also contain the same purchase order number and date. You should only denormalize your data model when it will enhance performance and simplify queries, or if you need to capture required historical information that would not otherwise be captured. When a database is denormalized, extra steps must be taken to ensure data integrity. In most cases, normalization is preferred.

You should not add multiple columns for **InvID** and **Quantity** values to the **PurchaseOrder** table and create a `FOREIGN KEY` constraint from each **InvID** column that references the **InvID** column in the **Inventory** table. You should not add multiple columns for the same attribute within an entity because this makes the data model less flexible and more difficult to query and maintain. For example, in this scenario, if you chose to add columns for each purchased item, each purchase order could only contain the number of items for which you added additional columns. If a purchase for more items was made, the table structure would have to be modified to include additional columns. If fewer items were purchased than there were columns for purchased items, some of the columns would be empty, thus unnecessarily increasing the size of the database. This data model would also make writing queries more difficult because you would have to query each purchase-related column.

QUESTION 137

You plan to create a Service Broker solution. The solution will transport data from one queue to another queue.

You need to identify which message type must be used to transport binary data. The solution must minimize the amount of data transported.

Which message type should you use?

A. EMPTY

B. NONE

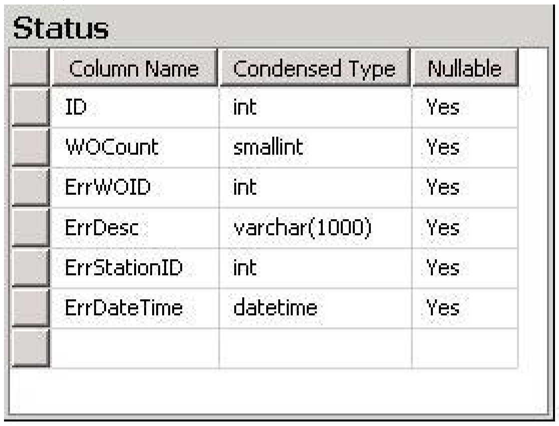
C. VALID_XML WITH SCHEMA COLLECTION

D. WELL_FORMED_XML

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 138
You are a database developer on an instance of SQL Server 2008. Your **Prod** database contains a **WorkOrderDetail** table that stores information
about work being processed within your facility.
As rows are inserted into the **WorkOrderDetail** table, you want to gather and record information in the **Status** table, which is defined as follows:



Status

Column Name	Condensed Type	Nullable
ID	int	Yes
WOCount	smallint	Yes
ErrWOID	int	Yes
ErrDesc	varchar(1000)	Yes
ErrStationID	int	Yes
ErrDateTime	datetime	Yes

If a row cannot be successfully added to the **WorkOrderDetail** table, three requirements must be met:
 A description of the error that occurred should be recorded in the **ErrDesc** column.
 The total number of work orders in progress should be recorded in the **WOCount** column.
 The date and time that an error occurred should be stored in the **ErrDateTime** column.
Which action should you take?

A. Enable Change Tracking for your **Prod** database.
B. Create a table-level `CHECK` constraint.
C. Create an `INSTEAD OF INSERT` trigger on the **WorkOrderDetail** table.
D. Create an `AFTER INSERT, UPDATE` trigger on the **WorkOrderDetail** table.

Answer: C
Section: (none)

Explanation/Reference:
An `INSTEAD OF` trigger fires
instead of the operation that fired the trigger. `INSTEAD OF` triggers can be used to prevent the original operation from taking place or to perform
additional or alternative actions. In this scenario, an `INSTEAD OF INSERT` trigger could include code to capture the additional information and
insert it into the **Status** table.
You should not create an `AFTER INSERT, UPDATE` trigger on the **WorkOrderDetail** table. An `AFTER` trigger only fires after the triggering
statement completes successfully. In this scenario, you needed to log errors that prevented rows from being inserted. This could not be performed
in an `AFTER` trigger.
You should not create a table-level `CHECK` constraint. A `CHECK` constraint is used to restrict the data that is allowed for a column to specific values.

A `CHECK` constraint consists of a Boolean expression that evaluates to either `TRUE` or `FALSE`. If the expression evaluates to `TRUE`, the value is
allowed for the column, and if the expression evaluates to `FALSE`, the value is not allowed for the column. `CHECK` constraints can be defined at the
table level or column level, but only `CHECK` constraints defined at the table level can use columns other than the constrained column in the
constraint expression.
You should not enable Change Tracking for your **Prod** database. The Change Tracking feature allows you to track rows in a table that were
modified by DML statements. Detailed information about the data that was modified is not tracked, but the fact that rows in the table were changed
is recorded in internal tables. This information can then be accessed using special change tracking functions.

QUESTION 139
You need to design a solution to guarantee that a stored procedure is part of every transaction.

A. Enable Implict transaction.
B. Set the NOCOUNT option to on.
C. Set the XACT_ABORT option to on.
D. Enable distributed transaction Cordinatoor (DTC).

Answer: A
Section: (none)

Explanation/Reference:

QUESTION 140
You are a database developer. You plan to design a database solution by using SQL Server 2008. You configure a database on a server to use a common language runtime (CLR). You need to create a CLR assembly that enables the CLR stored procedure to access environment variables available on the server. You also need to ensure that the CLR assembly has the minimum permissions assigned.What should you do?

A. Enable the TRUSTWORTHY database property.
B. Create the assembly by using the SAFE permission set.
C. Create the assembly by using the UNSAFE permission set.
D. Create the assembly by using the EXTERNAL ACCESS permission set.

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 141
You are a database developer. You develop solutions by using SQL Server 2008 in an enterprise environment. You are creating a SQL Agent job that uses Transact-SQL to update data in two related databases on two different servers.
You have the following requirements:
·
The job can only execute once each evening.
·

The databases on each server use the full-recovery model.
·
Transaction log backups for the two databases occur at different times.
·
The job uses transactions to ensure that in the event of an error, all updates are rolled back. You need to ensure that when you restore a database on either server, the two databases are restored to a state that reflects the last time the job successfully executed.
What should you do?

A. Ensure both databases are altered using the NO_WAIT termination clause.
B. Use the Windows Sync Manager to ensure that the databases can never be out of synchronization.
C. Use saved transactions. When a database failure occurs, restore both databases by using a saved transaction.
D. Use marked transactions. When a database failure occurs, restore both databases by using a marked transaction.

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 142
You are a database developer. You plan to design a database solution by using SQL Server 2008. The database will contain a common language runtime (CLR) user-defined scalar function. The function will return an integer value.
You need to ensure that the computed columns that use the result from this function can be indexed.
What should you do?

A. 1. Ensure that the logic of the function returns the same value for the same input values and the same database state.
   2. Ensure that the IsDeterministic property is set to True.
B. 1. Ensure that the logic of the function returns a different value for the same input values and the same database state.
   2. Ensure that the IsDeterministic property is set to True.
C. C. 1. Ensure that the logic of the function returns the same value for the same input values and the same database state.
   2. Ensure that the IsDeterministic property is set to False.
D. D. 1.·Ensure that the logic of the function returns a different value for the same input values and the same database state.
   2. Ensure that the IsDeterministic property is set to False.

Answer: A
Section: (none)

Explanation/Reference:

QUESTION 143
You are a database administrator on an instance of SQL Server 2008. You previously created a plan guide using the following Transact-SQL:
```
sp_create_plan_guide@name = N'SpecialInvTrxGuide',@stmt = N'SELECT * FROM
Inventory.TrxHistory AS h,Inventory.Product AS pWHERE h.ProductID = p.ProductID
```

```
AND TrxCategoryDesc = @CatDesc',@type = N'OBJECT',@module_or_batch = N'Inventory.
GetInventoryHistory',
@params = NULL,@hints = N'OPTION (OPTIMIZE FOR (@CatDesc = N''Material
Transaction''))'
```

You are evaluating the performance effectiveness of the plan guide. You want to disable the **SpecialInvTrxGuide** plan guide temporarily, but continue to use other plan guides that have been created. Which statement should you execute?

A. `EXEC sp_control_plan_guide N'DISABLE', N'SpecialInvTrxGuide'`

B. `EXEC sp_control_plan_guide N'DROP', N'SpecialInvTrxGuide'`

C. `ALTER GUIDE SpecialInvTrxGuide DISABLE;`

D.
```
sp_create_plan_guide @name = N'SpecialInvTrxGuide',@stmt = N'SELECT *FROM
Inventory.TrxHistory AS h,Inventory.Product AS pWHERE h.ProductID = p.ProductID
AND TrxCategoryDesc = @CatDesc',@type = N'NULL',@module_or_batch = N'Inventory.
GetInventoryHistory',@params = NULL,@hints = N'OPTION (OPTIMIZE FOR (@CatDesc =
N''Material Transaction''))'
```

Answer: A
Section: (none)

Explanation/Reference:
You should use the following statement:
`EXEC sp_control_plan_guide N'DISABLE', N'SpecialInvTrxGuide'`
The **sp_control_plan_guide** system stored procedure can be used to drop or disable a single plan guide or to drop or disable all plan guides in the current database. The procedure accepts two parameters. The first parameter is the action to be taken, and the second parameter is the plan guide name. You may specify a value of `DISABLE`, `DISABLE ALL`, `ENABLE`, `ENABLE ALL`, `DROP`, or `DROP ALL` for the first parameter. In this scenario, you wanted to disable a single plan guide. Therefore, you should call the procedure, passing it the value of `'DISABLE'` for the first parameter and the name of the plan guide as the second parameter. If you wanted to disable all the plan guides in the current database, you could include the `DISABLE ALL` value . You should not use the statement that calls the **sp_create_plan_guide** system stored procedure, passing an `@type` parameter value of `NULL`. This statement will generate the following syntax error because `NULL` is not a valid value for the `@type` parameter:

```
Msg 10501, Level 16, State 1, Procedure sp_create_plan_guide, Line 20Cannot
create plan guide 'SpecialInvTrxGuide' because type 'NULL' provided is not
allowed.
```

You should not use the `ALTER GUIDE` statement because this is not a valid SQL statement. This statement will generate the following syntax error:

```
Msg 343, Level 15, State 1, Line 2Unknown object type 'GUIDE' used in a CREATE,
DROP, or ALTER statement.
```

You should not use the statement that calls the **sp_control_plan_guide** system stored procedure with the `DROP` parameter value for the first parameter. This statement will permanently remove the plan guide from the database. In this scenario, you only wanted to disable the **SpecialInvTrxGuide** plan guide.

QUESTION 144
You are a database developer on an instance of SQL Server 2008. You created the **TrxHistory** table using the following statement:

```
CREATE TABLE Inventory.TrxHistory (ID int PRIMARY KEY,TrxID int,TrxDesc varchar
(50),TrxAmt money,TrxDate datetime,UpdID int);
```

The table has become extremely large and you want to minimize the storage it requires.
You want to determine the storage savings if you implemented page-level compression for the table.
Which statement should you use?

A.
```
EXEC sp_estimate_data_compression_savings 'Inventory', 'TrxHistory', NULL,
NULL, 'PAGE';
```

B. `EXEC sp_estimate_data_compression_savings 'Inventory.TrxHistory', 'PAGE';`
C. `EXEC sp_estimate_data_compression_savings 'Inventory', 'TrxHistory';`
D. `EXEC sp_spaceused N'Inventory.TrxHistory';`

Answer: A
Section: (none)

Explanation/Reference:
You can use the **sp_estimate_data_compression_savings** system stored procedure to estimate the space that would be saved by implementing row or page compression for a table. You can also use the procedure to estimate the result of disabling compression or implementing compression for an index or a single partition of a partitioned table. The procedure accepts the following input parameters:
`@schema_name`: Indicates the schema that contains the table. This parameter defaults to the current user's schema if a schema is not specified.
`@object_name`: Indicates the table name.
`@index_id`: Indicates the ID number of an index.
`@partition_number`: Indicates the partition number.
`@data_compression`: Indicates the type of compression. Valid values are `NONE`, `ROW`, and `PAGE`.

The procedure returns a column indicating the current size, **size_with_current_compression_setting**, and a column indicating the estimated size if the specified type of compression were implemented, **size_with_requested_compression_setting**. In this scenario, you would be able to determine the estimated savings if you implemented page-level compression for the **TrxHistory** table. You should note that when evaluating whether to implement compression, you should also consider how the data is accessed. Compression generates overhead, especially for tables that are frequently accessed. Therefore, both the storage savings and the performance impact should be considered.
You should not use either of the following statements:
```
EXEC sp_estimate_data_compression_savings 'Inventory.TrxHistory', 'PAGE';
EXEC sp_estimate_data_compression_savings 'Inventory', 'TrxHistory';
```
Both statements use invalid syntax, and would generate the following error:
```
Msg 201, Level 16, State 4, Procedure sp_estimate_data_compression_savings, Line
0Procedure or function
'sp_estimate_data_compression_savings' expects parameter '@index_id', which was
not supplied.
```
You should not use the following statement:
```
EXEC sp_spaceused N'Inventory.TrxHistory';
```
The **sp_spaceused** system stored procedure returns current space usage information, not estimated storage savings by implementing compression.

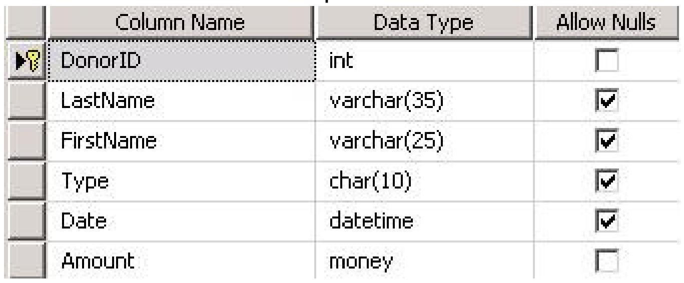
QUESTION 145
You plan to implement a Web-based application that will save XML data to a column in a table. You need to design a query that ensures that before saving the XML data to the table, the data contains valid elements. The solution must be developed by using the minimum amount of effort.
What should you include in the query?

A. .exist()
B. .query()
C. FOR XML PATH
D. sp_xml_preparedocument

Answer: A
Section: (none)

Explanation/Reference:

QUESTION 146
You are a database developer on an instance of SQL Server 2008. Your **Donors** table is defined as follows:



Column Name	Data Type	Allow Nulls
DonorID	int	☐
LastName	varchar(35)	✓
FirstName	varchar(25)	✓
Type	char(10)	✓
Date	datetime	✓
Amount	money	☐

You create a stored procedure using the following statement that accesses the **Donors** table:
```
CREATE PROCEDURE usp_GetNewDonors (@type char(10))
AS
SELECT DonorID,
FirstName,
LastName
FROM Donors
WHERE Type = @type;
```
Each time the stored procedure is called, you want it to execute with the security permissions of the current database user.
Which clause should you add to the `CREATE PROCEDURE` statement?

A. EXTERNAL NAME
B. EXECUTE AS SELF
C. EXECUTE AS OWNER
D. `WITH SCHEMABINDING`

Answer: B
Section: (none)

Explanation/Reference:

You should add the `EXECUTE AS SELF` clause to the stored procedure. This will allow the stored procedure to execute under the security context
of the current database user each time it is called. You can specify the following values in the `EXECUTE AS` clause:
 `SELF`: Specifies the stored procedure executes under the security context of the current user.
 `OWNER`: Specifies the stored procedure executes under the security context of the user that owns it.
 `CALLER`: Specifies the stored procedure executes under the security context of the user calling it. To execute the stored procedure
successfully, the user calling the stored procedure would require permissions on both the stored procedure and any underlying database
objects referenced by the stored procedure.
 `user_name`: Specifies the stored procedure executes under the security context of the specified user regardless of which user called the
stored procedure.
You should not add the `WITH SCHEMABINDING` clause to the stored procedure. The `WITH SCHEMABINDING` clause implements schema binding
for a view or function, but cannot be used with a stored procedure. Schema binding ensures that changes cannot be made to underlying objects that
might make a view or function unusable.
You should not add the `EXTERNAL NAME` clause to the stored procedure. The `EXTERNAL NAME` clause is

used when defining a CLR stored
procedure to identify which assembly method to execute.

QUESTION 147
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database which will be use by a scheduling application. The following information is stored in this database:
Rooms
Classes
Students
Teachers
You have to consider the following facts in the design plan:
Each room can host one or more classes;
Each class can be in one or more rooms; Each teacher can teach one or more classes; Each student can register for one or more classes.
Look at the entities below:
Rooms; Classes; Students; Teachers; ClassesTeachers; ClassesStudents You identify these entities for the database design. In order to ensure normalization, you have to design the database. So what should you do?

A. First, a new entity named ClassesRooms should be added. Then use this entity to establish a relationship between the Classes and Rooms entities
B. First, a new entity named TeachersRooms should be added. Then use this entity to establish a relationship between the Teachers and Rooms entities
C. First, a new entity named StudentsRooms should be created. Then use this entity to establish a relationship between the Students and Rooms entities.
D. First, a new entity named TeachersStudents should be added. Then use this entity to establish a relationship between the Teachers and Students entities

Answer: A
Section: (none)

Explanation/Reference:

QUESTION 148
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. There're two stored procedures in the database. The stored procedures are respectively named AlterSelling and GetSelling. Every day the two procedures are executed very often. The AlterSelling stored procedure updates a table named SellingOrders by using a transaction. From the SellingOrders table, data is retrieved and aggregated by the GetSelling stored procedure for a sales trend analysis report. When users run the sales trend analysis report, they have to wait for a long time.
So what should you do to make sure that sales trend analysis report runs as quickly as possible?

A. For AlterSelling, the isolation level should be changed to SERIALIZABLE
B. The NOLOCK hint has to be added to the SELECT statements in GetSelling
C. The NOWAIT hint has to be added to the SELECT statement in GetSelling
D. For AlterSelling, the isolation level should be changed to READ UNCOMITTED

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 149
Your company has the following development policy for XML data:
The data must be element-centric
The data must be well-formed XML
The data must have a root element
The data must contain the parent table name
You need to recommend guidelines for generating well-formed XML result sets.
What should you recommend?

A. FOR XML AUTO
B. FOR XML PATH
C. OPENXML()
D. XQUERY

Answer: B
Section: (none)

Explanation/Reference:

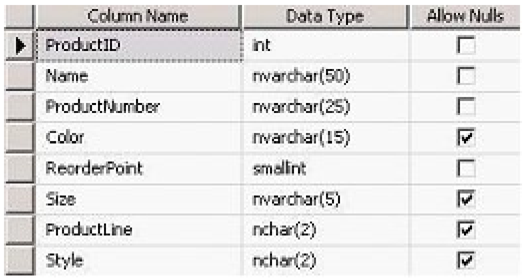
QUESTION 150
You have a table that has five varchar columns.
You are designing an application that requires data in well-formed XML. You need to design a query statement that will produce the data in well-formed XML.
What should you use?

A. FOR XML PATH
B. sp_xml_preparedocument
C. XPATH query
D. XSD schema

Answer: A
Section: (none)

Explanation/Reference:

QUESTION 151
You are a database developer on an instance of SQL Server 2008. You have a **ProductDetails** table defined as follows:



Column Name	Data Type	Allow Nulls
ProductID	int	☐
Name	nvarchar(50)	☐
ProductNumber	nvarchar(25)	☐
Color	nvarchar(15)	☑
ReorderPoint	smallint	☐
Size	nvarchar(5)	☑
ProductLine	nchar(2)	☑
Style	nchar(2)	☑

You want retrieve data from the table in the following XML format:

`<ProdData> <ProductDetails ID="1" Name="Adjustable Race" ProdNum="AR-5381" / ><ProductDetails ID="2" Name="Bearing Ball" ProdNum="BA-8327" /><ProductDetails ID="3" Name="BB Ball Bearing" ProdNum="BE-2349" /></ProdData>`

Which `FOR XML` clause should you use in your query?

A. `FOR XML RAW, ELEMENTS, ROOT('ProdData')`
B. `FOR XML AUTO, ELEMENTS, ROOT('ProdData')`
C. `FOR XML AUTO, ROOT('ProdData')`
D. `FOR XML PATH, ROOT('ProdData')`

Answer: C
Section: (none)

Explanation/Reference:
The `FOR XML` clause specifies that the result of the `SELECT` statement should be returned in XML format. You can specify one of the following modes with the `FOR XML` clause:
`RAW`: A single `<row>` element is generated for each row in the rowset. Each non-null column value generates an attribute with the name identical to the column's name or the column's alias.
`AUTO`: Nested elements are returned using the columns specified in the `SELECT` list. Each non-null column value generates an attribute that is named according to the column name or column alias. The element nesting is based on the order in which the columns are specified in the `SELECT` list.
`EXPLICIT`: Each row in the rowset can be defined in detail by specifying attributes and elements.
`PATH`: Each row in the rowset can be defined in detail, but column names and column aliases are specified as XPath expressions.

In this scenario, you wanted to retrieve an element for each row that contains an attribute for each referenced column and include a custom root element. When you use `AUTO` mode, the elements and attributes are generated based on their order in the `SELECT` list, and the attribute names will be created using column aliases. Specifying the `ROOT` option will add a root element with the given name. The following query could be used to generate the correctly formatted XML:

```
SELECT ProductID AS ID, Name, ProductNumber AS ProdNumFROM ProductDetails FOR XML
AUTO, ROOT('ProdData');
```

You should not use `FOR XML RAW, ELEMENTS, ROOT('ProdData')`. The `RAW` mode generates a single `<row>` element for each row. The `ELEMENTS` option indicates that you want to return columns in the `SELECT` list as subelements, rather than as attributes. The `ROOT` option adds a root element with the specified name. Using this clause in your query would produce output in the following format:

```
<ProdData>
<row> <ID>1</ID><Name>Adjustable Race</Name><ProdNum>AR-5381</ProdNum></row><row>
<ID>2</ID><Name>Bearing Ball</Name><ProdNum>BA-8327</ProdNum></row><row> <ID>3</
ID><Name>BB Ball Bearing</Name><ProdNum>BE-2349</ProdNum></row></ProdData>
```

You should not use `FOR XML AUTO, ELEMENTS, ROOT('ProdData')`. Using this clause in your query would produce XML output in the following format:

```
<ProdData> <ProductDetails> <ID>1</ID><Name>Adjustable Race</Name><ProdNum>AR-
5381</ProdNum></ProductDetails><ProductDetails> <ID>2</ID><Name>Bearing Ball</
Name><ProdNum>BA-8327</ProdNum></ProductDetails><ProductDetails> <ID>3</
ID><Name>BB Ball Bearing</Name><ProdNum>BE-2349</ProdNum></ProductDetails></
ProdData>
```

You should not use `FOR XML PATH(''), ROOT('ProdData')`. `PATH` mode allows you to define in detail the attributes and elements that should be generated in the XML output. The `ROOT` option will generate a root element with the specified name. Specifying `PATH('')` will cause all subsequent elements to be subelements of the root element. Using this clause in your query would produce output in the following format:

```
<ProdData> <ID>1</ID><Name>Adjustable Race</Name><ProdNum>AR-5381</
ProdNum><ID>4</ID><Name>Headset Ball Bearings</Name><ProdNum>BE-2908</ProdNum></
ProdData>
```

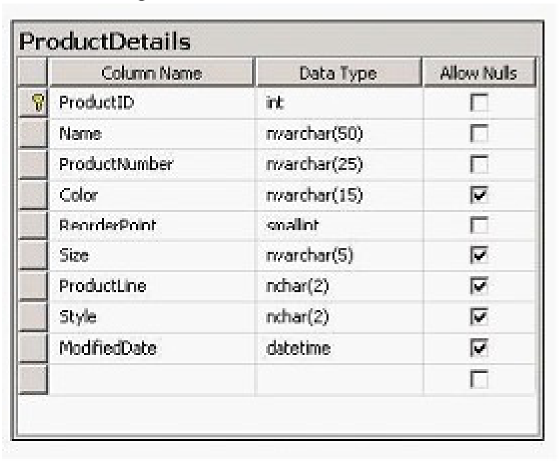
QUESTION 152
You have an instance of SQL Server 2008 that has xp_cmdshell enabled. You need to design a stored procedure that meets the following requirements:
·
Allows authorized users to retrieve lists of files
·
Minimizes permissions assigned to objects
·
Minimizes security risks
What should you include in the design?

A. Grant users permission to execute xp_cmdshell.
B. Grant users permission to execute sp_configure.
C. Create a procedure that uses EXECUTE AS OWNER. Call xp_cmdshell in the procedure. Grant users permission to execute the procedure.
D. Create a procedure that uses EXECUTE AS CALLER. Call xp_cmdshell in the procedure. Grant users permission to execute the procedure.

Answer: C
Section: (none)

Explanation/Reference:

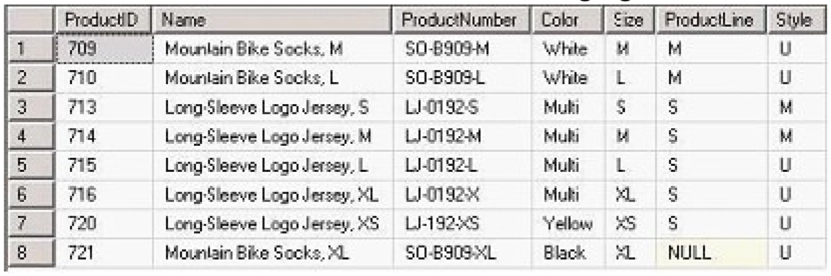
QUESTION 153
You are a database developer on an instance of SQL Server 2008. You have a **Prod** database that contains the following **ProductDetails** table:



ProductDetails

Column Name	Data Type	Allow Nulls
ProductID	int	☐
Name	nvarchar(50)	☐
ProductNumber	nvarchar(25)	☐
Color	nvarchar(15)	☑
ReorderPoint	smallint	☐
Size	nvarchar(5)	☑
ProductLine	nchar(2)	☑
Style	nchar(2)	☑
ModifiedDate	datetime	☑
		☐

You also have another table named **ProductStaging**, which contains the following data:

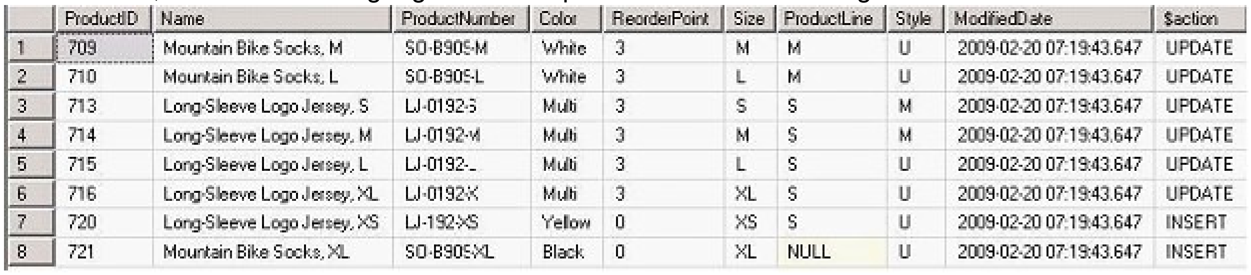


	ProductID	Name	ProductNumber	Color	Size	ProductLine	Style
1	709	Mountain Bike Socks, M	SO-B909-M	White	M	M	U
2	710	Mountain Bike Socks, L	SO-B909-L	White	L	M	U
3	713	Long-Sleeve Logo Jersey, S	LJ-0192-S	Multi	S	S	M
4	714	Long-Sleeve Logo Jersey, M	LJ-0192-M	Multi	M	S	M
5	715	Long-Sleeve Logo Jersey, L	LJ-0192-L	Multi	L	S	U
6	716	Long-Sleeve Logo Jersey, XL	LJ-0192-X	Multi	XL	S	U
7	720	Long-Sleeve Logo Jersey, XS	LJ-192-XS	Yellow	XS	S	U
8	721	Mountain Bike Socks, XL	SO-B909-XL	Black	XL	NULL	U

You execute the following statement:
```
MERGE ProductDetails AS t USING ProductStaging AS sON (t.ProductID = s.ProductID)
WHEN MATCHED THEN UPDATE SET ModifiedDate = GETDATE(),Color = s.Color,Style = s.
Style,ProductLine = s.ProductLine WHEN NOT MATCHED THEN INSERT(ProductID, Name,
ProductNumber, Color, ReorderPoint, Size, ProductLine, Style, ModifiedDate)VALUES
(s.ProductID, s.Name, s.ProductNumber, s.Color, 0, s.Size, s.ProductLine, s.
Style, GETDATE())OUTPUT INSERTED.*, $action;
```
What is the result?

A. If a row with a **ProductID** value of 720 does not exist in the **ProductDetails** table, the product is added to the **ProductDetails** table.
B. If a row with a **ProductID** value of 710 exists in the **ProductDetails** table, no action is taken.
C. If a row with a **ProductID** value of 800 exists in the **ProductDetails**, it is deleted.
D. If a row with a **ProductID** value of 713 exists in the **ProductDetails** table, it is updated only if the **Color**, **Style**, or **ProductLine** value is different.

Answer: A
Section: (none)

Explanation/Reference:
In this scenario, you issued a `MERGE` statement. The `MERGE` statement allows you to combine the inserts, deletes, and updates, and to use a single statement to perform multiple DML actions. Using a `MERGE` statement instead of issuing multiple DML statements can improve performance. In a `MERGE` statement, you specify a source and a target and include a join. Then, you use the `MATCHED` clauses to specify the actions to be performed. The basic syntax of the `MERGE` statement is as follows:
```
MERGE [INTO] target_table USING source_table ON join_condition [WHEN MATCHED
THENmatched_action][WHEN NOT MATCHED [BY TARGET] THEN notmatched_action][WHEN NOT
MATCHED BY SOURCE THEN notmatchedsource_action];
```
The `WHEN NOT MATCHED THEN` clause specifies the action to take if the records from the source table are not in the target table. The `WHEN MATCHED THEN` clause specifies the action to take if the records from the source table are in the target table. In this scenario, you specified an `UPDATE` statement in the `WHEN MATCHED THEN` clause. Therefore, if a row in the **ProductStaging** table has the same **ProductID** as a row in the **ProductDetails** table, the row will be updated with new values for the **ModifiedDate**, **Color**, **Style**, and **ProductLine** columns. In this scenario, you also included an `INSERT` statement in the `WHEN NOT MATCHED THEN` clause. If a row in the **ProductStaging** table does not have a **ProductID** that matches a row in the **ProductDetails** table, the row will be inserted with the current date as the **ModifiedDate** column value and a **ReorderPoint** value of 0.
In this scenario, you also included an `OUTPUT` clause. The `OUTPUT` clause allows you to retrieve and display information about the rows that were affected by the `MERGE` statement. The `OUTPUT` clause can display this information to the user, insert the data into another permanent or temporary table or table variable using an `INTO` clause, or pass the data to a nested DML statement for processing. Within the `OUTPUT` clause, you would specify the column values that should be retrieved by using the column names with the `INSERTED` and `DELETED` prefixes. The `DELETED` prefix returns the column value before the DML operation, and the `INSERTED` prefix returns the column value after the DML operation, but before executing any triggers.
You can also use `$action` to return a string indicating which type of DML operation affected the row. In this scenario, you specified `OUTPUT INSERTED.*, $action`. This would return a result set of the rows affected by the `MERGE`, with the target rows' values as they existed after the merge was performed. For example, in this scenario, the statement might generate output similar to the following:


	ProductID	Name	ProductNumber	Color	ReorderPoint	Size	ProductLine	Style	ModifiedDate	$action
1	709	Mountain Bike Socks, M	SO-B905-M	White	3	M	M	U	2009-02-20 07:19:43.647	UPDATE
2	710	Mountain Bike Socks, L	SO-B905-L	White	3	L	M	U	2009-02-20 07:19:43.647	UPDATE
3	713	Long-Sleeve Logo Jersey, S	LJ-0192-S	Multi	3	S	S	M	2009-02-20 07:19:43.647	UPDATE
4	714	Long-Sleeve Logo Jersey, M	LJ-0192-M	Multi	3	M	S	M	2009-02-20 07:19:43.647	UPDATE
5	715	Long-Sleeve Logo Jersey, L	LJ-0192-L	Multi	3	L	S	U	2009-02-20 07:19:43.647	UPDATE
6	716	Long-Sleeve Logo Jersey, XL	LJ-0192-X	Multi	3	XL	S	U	2009-02-20 07:19:43.647	UPDATE
7	720	Long-Sleeve Logo Jersey, XS	LJ-192-XS	Yellow	0	XS	S	U	2009-02-20 07:19:43.647	INSERT
8	721	Mountain Bike Socks, XL	SO-B905-XL	Black	0	XL	NULL	U	2009-02-20 07:19:43.647	INSERT

The option that states no action is taken if a row with a **ProductID** value of 710 exists in the **ProductDetails**

table is incorrect. In this scenario, you included a `WHEN MATCHED` clause that included an update to the **Color**, **Style**, **ProductLine**, and **ModifiedDate** columns if the source row is found in the target table. If a row with a **ProductID** value of 710 exists in the **ProductDetails** table, the row will be updated with new values for these columns.
The option that states a row with a **ProductID** value of 800 that exists in the **ProductDetails** will be deleted is incorrect. You can include `DELETE` statements within a `MERGE` statement. However, in this scenario, you did not specify any delete actions.
The option that states a row with a **ProductID** value of 713 that exists in the **ProductDetails** table is updated only if the **Color**, **Style**, or **ProductLine** value is different is incorrect. In this scenario, the row would be updated, even if it had the same values for the **Color**, **Style**, and **ProductLine** columns. The `UPDATE` statement would update the **ModifiedDate** column of the row with **ProductID** 713 in the **ProductDetails** table.

QUESTION 154
Your **Prod** database resides on an instance of SQL Server 2008. The **Item** table contains details for each item sold by your company. The **Item** table has an **ItemNo** column that is defined as the table's primary key.
You execute the following Transact-SQL:
```
CREATE PROCEDURE DeleteItem(@ItemNo int)AS
SAVE TRANSACTION ProcSave1;
BEGIN TRANSACTION;
BEGIN TRY DELETE FROM Item WHERE ItemNo = @ItemNo;END TRY BEGIN CATCH SELECT
ERROR_MESSAGE() AS ErrorMessage;
IF @@TRANCOUNT > 0ROLLBACK TRANSACTION;END CATCH;
IF @@TRANCOUNT > 0COMMIT TRANSACTION;
```
What is the result?

A. If the item passed as input to the procedure exists, it will be successfully deleted.
B. If the item passed as input does not exist, an error message will be displayed.
C. The code generates an error because you cannot use a `ROLLBACK TRANSACTION` statement within a `CATCH` block.
D. The code generates an error because you cannot create a savepoint outside a transaction.

Answer: D
Section: (none)

Explanation/Reference:
You can only create a savepoint within an active transaction. If you attempt to create a savepoint outside a transaction, the following error will occur:
```
Msg 628, Level 16, State 0, Procedure DeleteItem, Line 4Cannot issue SAVE
TRANSACTION when there is no active transaction.
```
Savepoints allow you to roll back portions of transactions as needed. To create a savepoint to which you can roll back, you use the `SAVE TRANSACTION` statement and specify a savepoint name. Then, when you issue a `ROLLBACK` statement, you can specify the savepoint to which you want to roll back. You can define multiple savepoints within a single transaction.
Because the code fails, the stored procedure is not created. Therefore, the options that state it executes are both incorrect.
The code does not generate an error because you cannot use a `ROLLBACK TRANSACTION` statement within a `CATCH` block. You can include a `ROLLBACK TRANSACTION` statement, with or without a specified savepoint, within a `CATCH` block. With a `TRY...CATCH` construct, statements that might generate an error are included in the `TRY` block, and exception-handling code is included in the `CATCH` block. If all the statements in the `TRY` block execute successfully with no errors, execution resumes immediately following the `CATCH` block. If an error occurs within the `TRY` block, then the `CATCH` block is executed. With a `TRY...CATCH` construct, you can also use built-in functions, such as `ERROR_MESSAGE` and `ERROR_SEVERITY`, within the `CATCH` block to return details about the error that occurred. Or, you can use the `RAISERROR` statement to return a specific custom error.

Suppose you used the following Transact-SQL to create a stored procedure:

```
CREATE PROCEDURE DeleteItem2(@ItemNo int)AS
BEGIN TRANSACTION;
INSERT INTO Item(ItemNo, Description)VALUES (4,'Generic Item');
SAVE TRANSACTION ProcSave1;
BEGIN TRY DELETE FROM Item WHERE ItemNo = @ItemNo;END TRY BEGIN CATCH ROLLBACK
TRANSACTION ProcSave1;END CATCH;
COMMIT TRANSACTION;
```

If you call **DeleteItem2** passing it a value that does not exist in the **Item** table, the first `INSERT` will insert a row. Then, within the `TRY` block, the attempt to delete the non-existent item will fail and control will pass to the `CATCH` block. Within the `CATCH` block, the `ROLLBACK TRANSACTION` statement will roll back the transaction to the **ProcSave1** savepoint. The insert will still be valid, and the `COMMIT TRANSACTION` statement will commit the insertion.
If you call the **DeleteItem2** stored procedure passing it a value that exists in the **Item** table, the row will be inserted, the specified row will be deleted, and the `CATCH` block will not execute.

QUESTION 155
You have to store windows media file (WMV) that has size greater that 2GB in the database.

What datatype should you chose?

A. image
B. filestream
C. varbinary(max)
D. file

Answer: B
Section: (none)

Explanation/Reference:
Only data stored as filestream can be over 2GB.

QUESTION 156
You are a database developer on an instance of SQL Server 2008. Your company has branch offices located in different geographic regions. Each branch office maintains a separate SQL Server instance that contains a **TransactionHistory** table, which contains a historical record of that branch's transactions. Each branch office's **TransactionHistory** table has the same table structure, except each table contains a `CHECK` constraint on the **BranchID** column that allows branch users to only add and update records for their specific branch.
You want to allow users at the corporate office to query transaction history from all branches. Most queries will access only transactions for a single branch office, but some queries will access transaction history across multiple branches based on a date range.
You want ensure that branch office users can only modify the appropriate data, but also provide optimum performance for queries.
Which action should you take?

A. Implement a nested view that exposes the required data.
B. Create a view for each branch office that includes the `WITH CHECK OPTION` clause.
C. Create a distributed partitioned view using the `UNION ALL` operator.
D. Create a single partitioned table that includes a single `CHECK` constraint and transaction history from all branch offices.

Answer: C

Section: (none)

Explanation/Reference:
Partitioned views are used when you have similar data stored in multiple tables and want to create a view to allow access to all of the data as if it were stored in a single table. The tables referenced in the view can reside on the same server or on different servers. Partitioned views are implemented using the `UNION ALL` operator. For example, if you had three separate tables with an identical structure on the same server, you might use the following statement to create a partitioned view that allows users to query data from all three tables:

```
CREATE VIEW PartView AS SELECT * FROM Table1 UNION ALL SELECT * FROM Table2 UNION
ALL SELECT * FROM Table3;
```

In this scenario, you have multiple **TransactionHistory** tables residing on separate servers, each of which contains a subset of transaction history data. You can create a view across servers, known as a distributed partitioned view, by first creating a linked server definition for each branch and then creating the partitioned view. The partitioned view is created using a fully-qualified name in each `SELECT` statement. Using a distributed partitioned view would allow branch offices to access their transactions and enforce the `CHECK` constraint defined for each branch's **TransactionHistory** table, but would also allow users at the corporate office to query all data as if it resided in a single table. In addition, if a corporate office user issued a query against one branch's transaction history, the query optimizer would use the individual `CHECK` constraints defined on the tables to optimize performance of the query and search only the required tables. Using partitioned views would also allow the base tables to be managed separately. This can improve availability and decentralize administration effort because each base table can be backed up, restored, reorganized, or managed individually as needed.
You should not implement a nested view that exposes the required data. A nested view is a view that references another view in its definition. A nested view would not be appropriate in this scenario. For optimum performance, you should avoid nested views when possible, and if you use nested views, you should limit the level of nesting where possible.
You should not create a view for each branch office that includes the `WITH CHECK OPTION` clause. If you created a view for each branch office, it would not maximize performance of corporate queries, and it would increase the complexity to create queries across all branches. A query that accessed transaction history across multiple branches would have to reference each branch's view. When creating a view, the `WITH CHECK OPTION` clause is used to ensure that no data modifications can be made through a view that would cause the underlying data to violate the view's definition.
You should not create a single partitioned table that includes a single `CHECK` constraint and transaction history from all branch offices. In this scenario, leaving the transaction history data in separate tables across multiple servers will allow transaction history at each branch to be managed independently. This will provide better data availability, while still providing optimum performance of corporate queries. You might choose to use a single partitioned table if all the base tables resided on the same server.

QUESTION 157
You are the database designer for a manufacturing company on an instance of SQL Server 2008. You are creating several stored procedures that will update tables in your database.
You want each stored procedure to return a custom error message to the user if any of the DML statements in the stored procedure fail. Each custom error message must include the name of the procedure in which the error occurred and the error's severity level.
Which action should you take to implement error handling in your stored procedures?

A. Implement an output parameter in each stored procedure to return the error message number.
B. Include a `TRY...CATCH` construct in each stored procedure and use the `RAISERROR` statement within each `CATCH` block.
C. Check the value of `@@ERROR` after each stored procedure call.
D. Add a custom error message for each stored procedure to the **sysmessages** table using the **sp_addmessage** system stored procedure.

Answer: B

Section: (none)

Explanation/Reference:
A `TRY...CATCH` construct can be used to catch execution errors with a severity level higher than 10, as long as the error does not end the database connection. Transact-SQL code that might generate an error is included in the `TRY` block. If an error with severity level greater than 10 that does not end the database connection occurs in the `TRY` block, control is transferred to the `CATCH` block. The `CATCH` block can contain Transact-SQL code to handle the error that occurred. With a `TRY...CATCH` construct, you can also use functions to retrieve additional information about an error, such as `ERROR_LINE`, `ERROR_MESSAGE`, `ERROR_PROCEDURE`, and `ERROR_SEVERITY`. The `RAISERROR` statement returns a user-defined error message and sets a flag to indicate that an error occurred. The full syntax of the `RAISERROR` statement is as follows:

```
RAISERROR ({msg_id | msg_str | @local_variable}{, severity, state}[, argument
[,...n]])[WITH [LOG] [NOWAIT] [SETERROR]];
```

The `RAISERROR` statement can return specific error message text or a custom error message that has been stored in the **sysmessages** table. To return an error message stored in **sysmessages**, you can pass the `RAISERROR` statement the error's **message_id** as the first parameter. In this scenario, you could include each DML statement within your stored procedures in a `TRY` block, and use the `RAISERROR` statement within the associated `CATCH` block to return the desired information about the error. You could use the `ERROR_PROCEDURE` function in the `CATCH` block to return the name of the procedure that generated the error and the `ERROR_SEVERITY` function to return the error's severity level.
You should not implement an output parameter in each stored procedure to return the error message number. In this scenario, some of your stored procedures execute multiple DML statements. Therefore, returning information for a single error is not sufficient.
You should not check the value of `@@ERROR` after each stored procedure call because the `@@ERROR` function returns the error number of only the most recently executed Transact-SQL statement. The `@@ERROR` function returns a value of zero if the most recently executed Transact-SQL statement ran without errors. If an error occurred that corresponds to an error stored in the **sysmessages** table, the function returns the **message_id** for the error. The value of `@@ERROR` is reset each time a new Transact-SQL statement is executed. Therefore, you should call the function immediately following the Transact-SQL statement for which you want to inspect the error number.
You should not add a custom error message for each stored procedure to the **sysmessages** table using the **sp_addmessage** system stored procedure. The **sp_addmessage** system stored procedure creates a custom error message and stores the new message in the **sysmessages** table in the **master** database. After a user-defined error message is created with **sp_addmessage**, Transact-SQL constructs or applications can reference it using the `RAISERROR` statement. In this scenario, adding custom error messages with the **sp_addmessage** system stored procedure would define custom error message, but not ensure these message were returned by the stored procedures. You would still need to add code within the stored procedures to implement error handling.

QUESTION 158
You have two offices, one in NY and one in Tokyo. The Main database is located in the NY office.

You must permit the Tokyo sales manager to produce local reports in order to reduce WAN traffic. Which option will best work?

A. implement Failover Clustering
B. implement replication with snapshot
C. implement a clustered index
D. Give the Tokyo manager reader only access

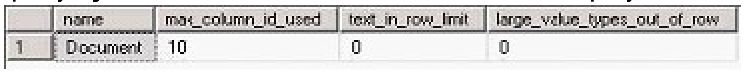
Answer: B
Section: (none)

Explanation/Reference:

QUESTION 159
You are a database administrator on an instance of SQL Server 2008. You issue the following statement to create the **Document** table:

```
CREATE TABLE Document(DocumentID int IDENTITY(1,1) NOT NULL,Title nvarchar](50)
NOT NULL,Status tinyint NOT NULL,DocumentSummary nvarchar(max) NULL,DocumentObj
varbinary(max) NULL);
```

You want to ensure that all **DocumentObj** values are stored in the database, but are not stored in-row. You query **sys.tables** for the **Document** table and display the following results:



	name	max_column_id_used	text_in_row_limit	large_value_types_out_of_row
1	Document	10	0	0

Which action should you take?

A. Implement page-level compression for the **Document** table.
B. Add the `FILESTREAM` attribute to the **DocumentObj** column.
C. Use **sp_tableoption** to set the **large value types out of row** option to `ON`.
D. Redefine the **DocumentObj** column as **text** data type.

Answer: C
Section: (none)

Explanation/Reference:
You should use **sp_tableoption** to set the **large value types out of row** option to `ON`. In this scenario, you want all values to be stored outside the data row. By default, SQL Server will attempt to store the data for large-value columns in-row. However, you can set the **large value types out of row** table option to `ON` to override this. For example, the following Transact-SQL will set the **large value types out of row** option to `ON` for the **Document** table:

```
EXEC sp_tableoption N'Document', 'large value types out of row', 'ON'
```

When this setting is `ON`, SQL Server will not attempt to store the data inside the data row.
You should not implement page-level compression for the **Document** table. Page-level compression may be used to save storage space, but does not determine whether data is stored in-row.
You should not add the `FILESTREAM` attribute to the **DocumentObj** column because this would cause the data to be stored on the file system, rather than in the database. FILESTREAM storage is implemented to store large binary objects, such as image or video files, as files on the file system and be able manage to them using Transact-SQL. FILESTREAM data can also be accessed using Win32 APIs. FILESTREAM storage is tightly integrated with most database functionality, including backup and recovery. When you take a database backup, FILESTREAM storage is also backed up unless you override this functionality by performing a partial backup. To create a table that can store FILESTREAM data, you create a table that contains a column of the **varbinary(max)** data type and include the `FILESTREAM` attribute.
You should not redefine the **DocumentObj** column as **text** data type. Both **text** and large value types, such as **varchar(max)**, **nvarchar(max)**, and **varbinary(max)**, can be stored in-row or outside the data row. The **text in row** option is used to control whether **text**, **ntext**, and **image** data types are stored in-row.

QUESTION 160
You have a database that has 20 large tables. All the tables have qualified indexes. As the tables grow, you discover that queries that contain JOIN statements execute more slowly. You need to recommend a solution to decrease the query response time and the I/O. The solution must minimize hardware costs.
What should you recommend?

A. Implement query hints.
B. Create a filegroup strategy.

C. Implement an index strategy.

D. Create a partitioning strategy.

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 161
You are a database developer. You plan to design a database solution by using SQL Server 2008. The database will contain three tables. The structure of the three tables is as shown in the following table.

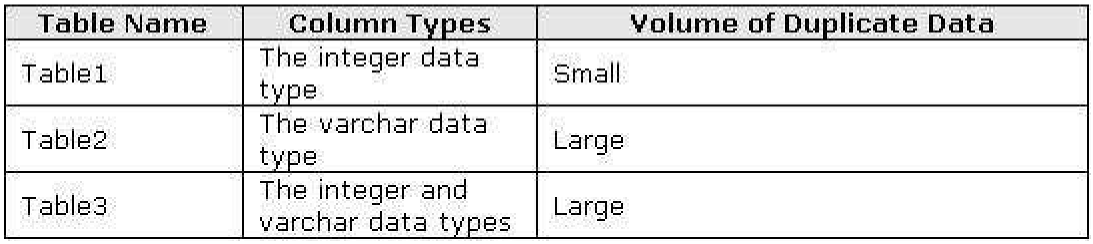


Table Name	Column Types	Volume of Duplicate Data
Table1	The integer data type	Small
Table2	The varchar data type	Large
Table3	The integer and varchar data types	Large

You need to minimize disk space usage without altering the data types in the tables of the database.
What should you do?

A. Implement row-level compression on all tables.

B. Implement row-level compression on Table1 and page-level compression on Table2 and Table3.

C. Implement row-level compression on Table2 and page-level compression on Table1 and Table3.

D. Implement row-level compression on Table3 and page-level compression on Table1 and Table2.

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 162
You are a database developer on an instance of SQL Server 2008. You are designing an application to be used by the human resources
department. Your **Employee** table needs to contain the following information:
 The employee's unique identifier
 The employee's last name and first name
 The numeric identifier of the department in which the employee works
 The unique employee identifier of the employee's manager
 The date the employee was hired
You want to be able to easily navigate the organization chart using a tree-like structure and easily identify each employee's level in the organization
chart. You also want to minimize code complexity. Which action should you take?

A. Implement separate tables for the employees and the employees' managers with a `FOREIGN KEY` constraint that relates the two tables.

B. Implement a single **Employee** table with the **EmpID** column as a **hierarchyid** data type.

C. Implement a single **Employee** table that represents each employee using an **xml** data type.
D. Implement a single **Employee** table that includes a `FOREIGN KEY` constraint on the **MgrID** column that references the **EmpID** column.

Answer: B
Section: (none)

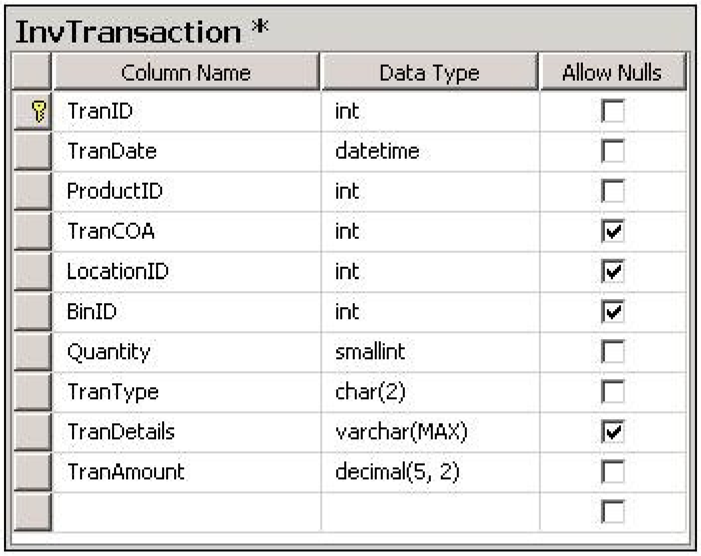
Explanation/Reference:
The **hierarchyid** data type is a special
variable-length, CLR-supplied data type that can be used to represent hierarchical data. When you define a **hierarchyid** data type column, you can
use various system-supplied methods, such as **GetRoot**, **GetDescendant**, **GetLevel**, **Read**, and **Write**, which allow you to perform tasks on the
hierarchy. Each value stored in the table is an internal binary value that represents the row's position in the hierarchy. In this scenario, you could
use the **GetLevel** method to easily return an employee's level in the organization chart.
Then, you could use statements similar to the following to perform tasks in the hierarchy:

```
-- Insert an employee into the table at the root node of the hierarchy
-- GetRoot returns the appropriate hierarchyid to store in the HierID column
INSERT INTO dbo.Employee (EmpID, HierID, LastName, FirstName, DeptID, HireDate,
Salary)
VALUES(1, hierarchyid::GetRoot(), 'Jones', 'Cindy', 45, GETDATE(), 97000.00);
GO
-- Insert an employee that is a subordinate to the root
-- GetRoot returns the hierarchyid of the root
-- GetDescendant returns the hierarchyid of the new employee's manager
DECLARE @parent hierarchyid
SELECT @parent = hierarchyid::GetRoot()
INSERT INTO dbo.Employee (EmpID, HierID, LastName, FirstName, DeptID, HireDate,
Salary)
VALUES (2, @parent.GetDescendant(NULL, NULL), 'King', 'Scott', 45, GETDATE(),
52000.00)
GO
-- Insert an employee that is a subordinate to EmpID 2
DECLARE @parent hierarchyid, @child hierarchyid
SELECT @parent = HierID FROM Employee WHERE EmpID = 2
INSERT INTO dbo.Employee (EmpID, HierID, LastName, FirstName, DeptID, HireDate,
Salary)
VALUES (3, @parent.GetDescendant(NULL, NULL), ' Rodriguez', 'Raymond', 45,
GETDATE(), 37000.00)
GO
-- Query all employees including a string that represents the path in the
hierarchy
SELECT HierID.ToString() AS 'HierString', *
FROM dbo.Employee
GO
```

The **ToString** method in the final `SELECT` statement converts the employee's hierarchical value to a string. The root employee will have a
HierString value of '/'. The employee with an **EmpID** value of 3 has a **HierString** value of '/1/1/'.
You should not implement separate tables for the employees and the employees' managers with a `FOREIGN KEY` constraint that relates the two
tables, or implement a single **Employee** table that includes a `FOREIGN KEY` constraint on the **MgrID** column that references the **EmpID** column. In
this scenario, using a **hierarchyid** data type would provide the desired functionality and be less complex to implement. If you implemented separate
tables for the employees and the employees' managers, you would have to combine datasets each time you wanted to query all employees. If you
implemented a single **Employee** table that included a `FOREIGN KEY` constraint on the **MgrID** column that

references the **EmpID** column, it would
require a single join to return an employee's manager, but determining the employee's level in the organization chart would be extremely complex to
code.
You should not implement a single **Employee** table that represents each employee using an **xml** data type. An **xml** data type is used to store XML
data, such as XML documents or XML document fragments.

QUESTION 163
You are a database developer on an instance of SQL Server 2008. You have a database that contains a large table named **InvTransaction**.
The **InvTransaction** table contains historical data on inventory transactions that previously occurred. The table is queried occasionally and used to
produce monthly reports, but is very rarely modified.
The **InvTransaction** table is defined as follows:



InvTransaction *

Column Name	Data Type	Allow Nulls
TranID	int	☐
TranDate	datetime	☐
ProductID	int	☐
TranCOA	int	✓
LocationID	int	✓
BinID	int	✓
Quantity	smallint	☐
TranType	char(2)	☐
TranDetails	varchar(MAX)	✓
TranAmount	decimal(5, 2)	☐
		☐

The **InvTransaction** table contains millions of rows, and has the following additional characteristics:
- A clustered index exists on the **TranID** column.
- The **LocationID** column for most transactions has a value of 1.
- The **TranType** column for most transactions has a value of 'T1'.
- Each **BinID** column has a bin value between 1 and 100 because there are a maximum of 100 bins for each location.
- Most of the **TranDetails** column values have similar descriptions.

You want to minimize the amount of storage used by the table, but not directly modify any of the table's columns.
Which action should you take?

A. Implement sparse columns for the **LocationID**, **TranType**, and **BinID** columns.
B. Set the **text in row** option to `OFF` for the **InvTransaction** table.
C. Issue the `ALTER TABLE REBUILD` statement with the `DATA_COMPRESSION=PAGE` option for the **InvTransaction** table.
D. Issue the `ALTER TABLE REBUILD` statement with the `DATA_COMPRESSION=ROW` option for the **InvTransaction** table.

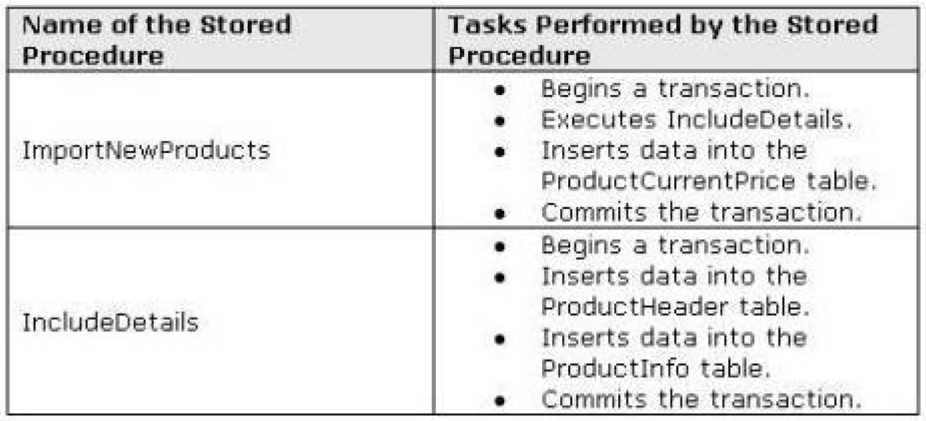
Answer: C
Section: (none)

Explanation/Reference:
You should issue the `ALTER TABLE REBUILD` statement with the `DATA_COMPRESSION=PAGE` option for the **InvTransaction** table. In this
scenario, you have rows with **LocationID**, **TranType**, and **BinID** values that are small. Therefore, you should implement row compression. Row
compression saves storage space for each row by internally compressing each fixed-length numeric, date/time, or character data type. For
example, if you implemented row compression for the **InvTransaction** table, the **LocationID** and **BinID** columns would use only the space
required, rather than using the four bytes that an **int** data type requires. Because most rows have a **LocationID** value of 1 and a **TranType** value
between 1 and 100, three bytes per column would be saved for most rows in the table.
In this scenario, you also have data that is frequently duplicated within the **LocationID**, **TranType**, **BinID**, and **TranDetails** columns. Therefore, you
should also implement page compression. Page compression is implemented internally by SQL Server to minimize the storage required to store
duplicated data. SQL Server uses a column-prefix or page-dictionary compression technique to eliminate redundant data within each page.
When you implement page compression, SQL Server implements also implements row compression. Therefore, in this scenario, rebuilding the
table with page-level compression would perform both types of compression and minimize storage for the **InvTransaction** table. The syntax of
rebuilding a table with compression is as follows:

```
ALTER TABLE table_name
REBUILD WITH (DATA_COMPRESSION = {NONE | ROW | PAGE});
```

You should not implement sparse columns for the **LocationID**, **TranType**, and **BinID** columns. When creating a table, sparse columns can be used
to optimize the storage of `NULL` values and minimize storage requirements. This would not be applicable in this scenario because there were no
columns mentioned that contained mostly `NULL` values. In addition, you did not want to modify any of the table's columns directly.
You should not set the **text in row** option to `OFF` for the **InvTransaction** table. The **text in row** option is used to control whether **text**, **ntext**, or
image columns are stored in the data row of the table. You could store data out-of-row to minimize storage used by the table. But in this scenario,
setting this option would not be appropriate because you do not have any **text**, **ntext**, or **image** columns in the **InvTransaction** table.
You can also use the **large value types out of row** option to specify whether large value types, such as **varchar(max)**, **nvarchar(max)**, and
varbinary(max), are stored in the data row.
You should not issue the `ALTER TABLE REBUILD` statement with the `DATA_COMPRESSION=ROW` option for the **InvTransaction** table. This would
only implement row compression for the table. In this scenario, implementing page compression would further minimize storage requirements.

QUESTION 164
There're two stored procedures in an application. The exhibit below shows the tasks performed by the stored procedures.



Name of the Stored Procedure	Tasks Performed by the Stored Procedure
ImportNewProducts	• Begins a transaction. • Executes IncludeDetails. • Inserts data into the ProductCurrentPrice table. • Commits the transaction.
IncludeDetails	• Begins a transaction. • Inserts data into the ProductHeader table. • Inserts data into the ProductInfo table. • Commits the transaction.

Sometimes foreign key violation errors are thrown by the procedures.

When ImportNewProducts inserts records into the ProductCurrentPrice table, it throws an error.
When ImportNewProducts inserts records into the ProductInfo table, it throws an error.

You must make sure that the two requirements below can be met:
- All transactions are rolled back if an error occurs in the INSERT statement of ProductCurrentPrice.
- Records inserted into ProductHeader and ProductCurrentPrice are committed if an error occurs in the INSERT statement of ProductCurrentPrice.

So what should you do to ensure this?

A. In ImportNewProducts, add a SET XACT_ABORT OFF statement;in IncludeDetails, add a SET XACT_ABORT ON statement.
B. In ImportNewProducts, add a SET XACT_ABORT ON statement;in IncludeDetails, add a SET XACT_ABORT OFF statement.
C. In a TRY/CATCH block, enclose all statements of ImportNewProducts In the CATCH block add aROLLBACK TRANSACTION statement.
D. In a TRY/CATCH block, enclose all statements of IncludeDetails In the CATCH block, add aROLLBACK TRANSACTION statement.

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 165
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. A database will contain 10 tables. Reports are generated by using the tables. The range of the data in the tables varies from 50,000 to 100,000 records. In the process that a query joins four tables, the problems below occur:
It takes a quite long time for the query to execute. And besides this, the size of the tempdb database grows quickly. Now you have to analyze the execution plan to check the cause for the two problems. So what should you do to find out the most possible cause for the problems?

A. In the execution plan, you look for table scans
B. In the execution plan, you look for Hash Match operators

C. In the execution plan, you look for Merge Join operators
D. In the execution plan, you look for Nested Loops operators

Answer: B
Section: (none)

Explanation/Reference:

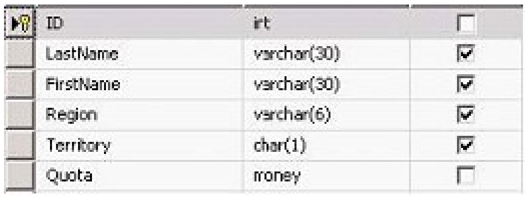
QUESTION 166
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. A stored procedure that uses the TRY/CATCH syntax is created in a new database. When the stored procedure is executed, the store procedure logs information about each step in the TRY block into a table when it is executed, the table is named dbo.RunningLog. If an error happens, the stored procedure must roll back the changes made to the target tables and retain the log entries stored in the dbo.RunningLog table. So what should you do to make sure that the given tasks can be performed by the stored procedure?

A. In the TRY block, you should start a transaction. Define a transactional save point before each step. Insert log entries into the dbo.RunningLog table after each step. Roll back to the transactional save points in the CATCH block. Commit the transaction after the CATCH block.
B. In the TRY block, you should start a transaction in the TRY block. Insert log entries into the dbo. RunningLog table after each step. Commit the transaction in the CATCH block after the CATCH block. Use data in the dbo.RunningLog table to reverse any changes made to the target tables. Commit the transaction if one exists
C. Before the TRY block, you define a table variable by using the same columns as that of the dbo. RunningLog table. In the TRY block, start a transaction. After each step, insert log entries Certkey.com - Make You Succeed To Pass IT Exams
Certkey 70-451
into the table variable. Roll back the transaction in the CATCH block. After the CATCH block, insert the rows from the table variable into the dbo.RunningLog table. Commit the transaction if one exists
D. Before the TRY block, you should use the same columns as that of the dbo.RunningLog table to define a temporary table by. In the TRY block, start a transaction. After each step, insert log entries into the temporary table. In the CATCH block, roll back the transaction. Insert the rows from the temporary table into the dbo.RunningLog table after the CATCH block.
Commit the transaction if one exists.

Answer: C
Section: (none)

Explanation/Reference:

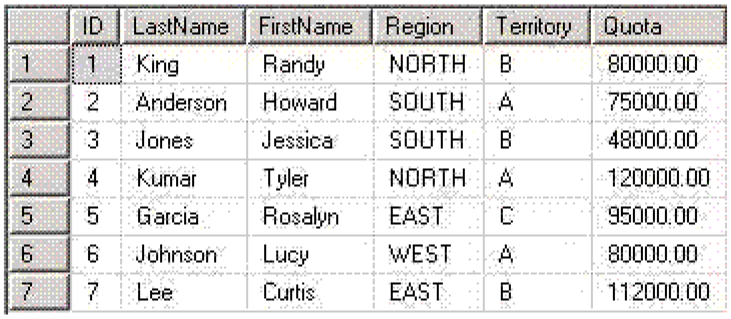
QUESTION 167
You are a database developer on an instance of SQL Server 2008. Your **Prod** database contains a **Salesperson** table that contains information about your company's salespeople. The **Salesperson** table is defined as follows:



ID	int	☐
LastName	varchar(30)	☑
FirstName	varchar(30)	☑
Region	varchar(6)	☑
Territory	char(1)	☑
Quota	money	☐

The **Salesperson** table contains the data shown in the exhibit. (Click the **Exhibit(s)** button.)
You want to create a query using the **Salesperson** table. You want the query to return a ranked list of salespeople by region. Within each region, salespeople should be ranked from highest to lowest based on their assigned sales quotas, with the same ranking given to salespeople who have the same assigned quota. You want the query provide optimum performance.
Which action should you take?

Exhibit:

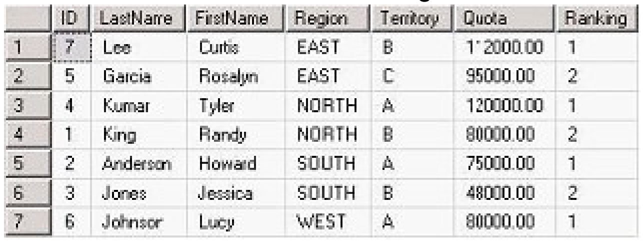


	ID	LastName	FirstName	Region	Territory	Quota
1	1	King	Randy	NORTH	B	80000.00
2	2	Anderson	Howard	SOUTH	A	75000.00
3	3	Jones	Jessica	SOUTH	B	48000.00
4	4	Kumar	Tyler	NORTH	A	120000.00
5	5	Garcia	Rosalyn	EAST	C	95000.00
6	6	Johnson	Lucy	WEST	A	80000.00
7	7	Lee	Curtis	EAST	B	112000.00

A. Include a `RANK` function in the `SELECT` list and a correlated subquery in the `FROM` clause.
B. Include an `OUTER APPLY` clause.
C. Include a `CROSS APPLY` clause.
D. Include a `DENSE_RANK` function with an `OVER` clause in the `SELECT` list.

Answer: D
Section: (none)

Explanation/Reference:
The `DENSE_RANK` function ranks rows based on the `PARTITION BY` and `ORDER BY` specified in the `OVER` clause. The basic syntax of the `DENSE_RANK` function is as follows:
```
DENSE_RANK ( ) OVER ([partition_by] order_by)
```
First, if a `partitioned_by` value is specified, the rows are partitioned. Then, within each partition, the rows are sorted and ranked based on the specified `ORDER BY` clause. If two rows within the same partition have the same `ORDER BY` value, they are assigned the same ranking, and the following ranked row is assigned the next sequential ranking value. For example, the following `SELECT` statement could be used in this scenario:
```
SELECT *, DENSE_RANK()OVER(PARTITION BY Region ORDER BY Quota DESC) AS
RankingFROM Salesperson;
```
This statement would partition the salespeople by region. Then, within each region, the rows would be sorted in descending order based on each salesperson's**Quota** value. Finally, the **Ranking** value would be calculated for each salesperson. Salespeople with identical **Quota** values within a region would have the same ranking, and no ranking values would be skipped. With the data given for the **Salesperson** table, this statement would return the following result:



	ID	LastName	FirstName	Region	Territory	Quota	Ranking
1	7	Lee	Curtis	EAST	B	1'2000.00	1
2	5	Garcia	Rosalyn	EAST	C	95000.00	2
3	4	Kumar	Tyler	NORTH	A	120000.00	1
4	1	King	Randy	NORTH	B	80000.00	2
5	2	Anderson	Howard	SOUTH	A	75000.00	1
6	3	Jones	Jessica	SOUTH	B	48000.00	2
7	6	Johnsor	Lucy	WEST	A	80000.00	1

You should not include a `RANK` function in the `SELECT` list and a correlated subquery in the `FROM` clause because a subquery is not needed in this scenario. You can use the`RANK` function to rank a result set. A correlated subquery is a subquery that references one or more columns in the outer query. Correlated subqueries can adversely affect performance, and should be avoided when possible because the inner query will execute once for each row of the outer query. While correlated subqueries are required in some situations, in this scenario you could accomplish the desired result using the aggregate `DENSE_RANK` function with an

OVER clause. While the RANK function is similar to the DENSE_RANK function, it skips a ranking value for each row that has an identical ORDER BY value.
You should not include an OUTER APPLY or a CROSS APPLY clause. The APPLY clause is used in the FROM clause of a query to join a table to a table-valued function. The table-valued function is called for each row returned by the outer query. The APPLY clause allows you to easily call a table-valued function for each row returned by a query. The OUTER APPLY clause returns all rows from the outer query, even if the row does not return a value for the table-valued function. The CROSS APPLY clause returns only the outer query rows for which the table-valued function returned a value.

QUESTION 168
You are a database developer on an instance of SQL Server 2008. Your **Transaction** table has a PRIMARY KEY constraint defined on the **TrxID** column. Your **Transaction** table also contains columns named **TrxType** and **TrxDisposition**.
You want to generate a list of the number of transactions by type and disposition. You want the result set to display the following data:
A total transaction count for each type
A total transaction count for each disposition
A grand total count of all transactions

Which action should you take?

A. Include only the **TrxID** column in the GROUP BY clause and specify a grouping set that includes only an empty set.
B. Include only the **TrxType** column in your GROUP BY clause and create a grouping set that includes the **TrxType** column.
C. Include the **TrxType** and **TrxDisposition** columns in your GROUP BY clause and include the WITH ROLLUP option.
D. Include the **TrxType** and **TrxDisposition** columns in your GROUP BY clause and include the WITH CUBE option.

Answer: D
Section: (none)

Explanation/Reference:
. The WITH CUBE clause is used with a GROUP BY clause to return rows that contain summary information. When you specify WITH CUBE, a summary row is included in the result set for each possible combination of the columns specified in the GROUP BY clause. For example, assume you executed the following statement:
```
SELECT TrxType, TrxDisposition, COUNT(TrxID) AS TrxCountFROM dbo.Transaction
GROUP BY TrxType, TrxDisposition WITH CUBE;
```
The summary rows that represent totals for each transaction type would have a transaction type in the **TrxType** column, and a NULL value in the **TrxDisposition** column. The summary rows that represent totals for each disposition would have a disposition in the **TrxDisposition** column, and a NULL value in the **TrxType** column. The grand total summary rows would have a NULL value for both the **TrxType** and **TrxDisposition** columns.
You should not include only the **TrxID** column in the GROUP BY clause and specify a grouping set that includes only an empty set, nor include only the **TrxType** column in your GROUP BY clause and create a grouping set that includes the **TrxType** column. Neither of these approaches would generate the desired result. You can use grouping sets to explicitly specify the groups for which aggregate information should displayed. Grouping sets are specified by including the GROUPING SETS clause with the GROUP BY clause. The GROUPING SETS clause allows you to explicitly specify the groups for which aggregate information should be displayed. This allows you to use more than one grouping within a single query. The syntax of the GROUP BY clause with a GROUPING SETS clause is as follows:
```
GROUP BY GROUPING SETS (groupingset1 [,...groupingsetn])
```
Each grouping set specified can contain one or more columns or an empty set, specified as (). Aggregate

rows are returned in the result set for only the specified groups. Specifying an empty set indicates that a grand total row should also be returned in the result set.
You should not include the **TrxType** and **TrxDisposition** columns in your `GROUP BY` clause and include the `WITH ROLLUP` option. Including the `WITH ROLLUP` clause would display the total number of transactions of each type and grand totals, but not the number of transactions for each disposition.

QUESTION 169
You maintain a database named **Manufacturing** on an instance of SQL Server 2008.
You have created a stored procedure named **usp_WorkOrderUpdate** that is frequently used to update the status of work orders as they are completed. The **usp_WorkOrderUpdate** stored procedure performs related updates to several tables in the database when a work order is marked as complete. You have used transactions within the procedure to ensure that all table updates are successful.
You need to issue a query against the **WorkOrderHistory** table, which is one of the tables updated in the **usp_WorkOrderUpdate** stored procedure. You want your query to execute as quickly as possible, ignoring any locks that the **usp_WorkOrderUpdate** procedure has acquired, and reading all rows even if they contain uncommitted data.
Which action should you take?

A. Include the `NOLOCK` table hint in your query.
B. Include the `READPAST` table hint in your query.
C. Modify the transaction isolation level for **usp_WorkOrderUpdate**.
D. Set `LOCK_TIMEOUT` to 0 before executing your query.

Answer: A
Section: (none)

Explanation/Reference:
When you include the `NOLOCK` table hint in a query, any locks currently held on the table are ignored. This will allow your query to execute more quickly, but the query will allow dirty reads. The query would return rows that the **usp_WorkOrderUpdate** stored procedure has updated but not yet committed. For example, in this scenario, the following query could be used to query the **WorkOrderHistory** table and ignore any locks currently held on the table:

```
SELECT * FROM WorkOrderHistory (NOLOCK);
```

You can specify separate locking hints for each table if a query joins multiple tables, and only ignore locks on specific tables in the query. You should note that the `NOLOCK` hint can only be used with `SELECT` statements. If you attempt to include a `NOLOCK` hint in an `INSERT`, `UPDATE`, `DELETE`, or `MERGE` statement, an error similar to the following occurs:

```
Msg 1065, Level 15, State 1, Line 15The NOLOCK and READUNCOMMITTED lock hints are
not allowed for target tables of INSERT, UPDATE, DELETE or MERGE statements.
```

You should not include the `READPAST` table hint in your query because in this scenario, you wanted to read uncommitted data. The `READPAST` hint will continue to process rows, but will not return any locked rows. This will allow the query to execute quickly, but the query will bypass any rows that **usp_WorkOrderUpdate** has modified but not yet committed.
You should not modify the transaction isolation level for **usp_WorkOrderUpdate** because this might adversely affect the modifications made by the **usp_WorkOrderUpdate** stored procedure.
You should not set `LOCK_TIMEOUT` to 0 before executing your query. The `LOCK_TIMEOUT` setting specifies how long a SQL statement should wait for a lock to be released before it returns an error. If you set `LOCK_TIMEOUT` to 0, the query would immediately return the following error if the **usp_WorkOrderUpdate** stored procedure had rows locked:

```
Msg 1222, Level 16, State 45, Line 1Lock request time out period exceeded.
```

You can use the `@@LOCK_TIMEOUT` function to determine the current `LOCK_TIMEOUT` setting. Including the `NOWAIT` hint would have the same result as setting the `LOCK_TIMEOUT` to 0.

QUESTION 170
You intend to use SQL Server 2008 to create a database solution. There's a database that contains a table. Besides this, it also contains a table-valued function which accepts the primary key from the table as a parameter. You intend to write a query. The query joins the table to the results of the table-valued function.

You must make sure that rows from the table returned are only those produce a result set from the table-valued function. So which join predicate should you choose to use?

A. INNER JOIN
B. CROSS APPLY
C. OUTER APPLY
D. LEFT OUTER JOIN

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 171
You are a database developer. You plan to design a database solution by using SQL Server 2008.
A database will contain 10 tables that are used to generate reports. Data in the tables ranges from 50,000 to 100,000 records.
During a query execution that joins four tables, you discover the following problems:
·The size of the tempdb database grows considerably.
·The query execution time is excessive.
You need to identify the most likely cause for the problems by analyzing the execution plan.
What should you do?

A. Look for table scans in the execution plan.
B. Look for Merge Join operators in the execution plan.
C. Look for Hash Match operators in the execution plan.
D. Look for Nested Loops operators in the execution plan.

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 172
You are a database developer on an instance of SQL Server 2008. You have an **EventMaster** table defined using the following statement:
```
CREATE TABLE EventMaster (EventID int PRIMARY KEY,EventDesc varchar(50) NOT NULL,
Provider varchar(30),EventDate datetime,Cost money,MaxSeats int);
```
You currently have a clustered index on the table's primary key, and a nonclustered index on the **Provider** column.
Rows are frequently added to the **EventMaster** table as new events are scheduled. Your booking representatives frequently query the **EventMaster** table to return the event description and cost of events provided by specific providers within a given time range. You want to optimize query performance.
Which action should you take?

A. Modify the nonclustered index to include **EventDate**.

B. Create a filtered index on **Provider** and **EventDate**.
C. Create an indexed view on **EventMaster**.
D. Create a clustered index on **EventDesc**.

Answer: A
Section: (none)

Explanation/Reference:
To improve query performance, indexes should be created on the columns that are frequently referenced in a `WHERE` clause. In this scenario, queries are executed to search for the events offered by specific providers during specific date ranges. Therefore, a composite nonclustered index on these two columns is likely to improve query performance.
You should not create a clustered index on **EventDesc** because the **EventMaster** table already contains a clustered index. A clustered index defines the order in which rows in a table are physically stored. Each table can have only one clustered index. When defining a primary key, SQL Server automatically creates a clustered index on the primary key column or columns if a clustered index does not exist, unless you specify otherwise. In this scenario, a clustered index already exists on the **EventID** column. Therefore, you cannot create another clustered index on the **EventDesc** column.
You should not create an indexed view on the **EventMaster** table because the scenario did not state that a view was defined. An indexed view is created by creating a unique clustered index on a view. This can often increase the view's performance. In a view with a clustered index, the result set is stored in the database, similar to storing a table with a clustered index. This reduces the overhead of generating a result set each time the view is referenced.
You should not create a filtered index on **Provider** and **EventDate**. A filtered index is a nonclustered index that is defined including a `WHERE` clause to optimize the index for queries that access specific subsets of the data.

QUESTION 173
Certkey.com - Make You Succeed To Pass IT Exams
Certkey 70-451

You are a database developer. You plan to design a database solution by using SQL Server 2008.
The database has a table named Sales. The Sales table contains 10 million rows. You discover that the following query takes a long time to execute.
SELECT s.sale_id, ...
FROM Sales AS s
JOIN Country AS c
ON s.Country_id = c.Country_id
AND c.Country_name = 'USA'
A summary of the execution plan is as shown in the following code segment. |--Hash Match(Inner Join, HASH:([s].[Country_id]) = ([c].[Country_id]) |--Clustered Index Scan(OBJECT:([Country].[PK_Country_Country_id] AS [c]) |--Clustered Index Scan(OBJECT:([Sales].[PK_Sales_Sale_id] AS [s])) You need to ensure that the query retrieves data in minimum possible time.
What should you do?

A. Modify the query to use a loop join hint.
B. Modify the query to use a merge join hint.
C. Create a nonclustered index in the Country_id column of the Sales table.
D. Create a nonclustered index in the Country_name column of the Country table.

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 174
You are a database developer. You develop a database by using SQL Server 2008 in an enterprise environment.

The database has a table named Sales.Inventory. The table is partitioned into four geographic regions. You update the Sales.Inventory table for each region by using the following stored procedure.

```
CREATE STORED PROCEDURE usp_Update
@RegionID tinyint
AS
UPDATE Sales.Inventory
SET Qty = T.CurrentQuantity
FROM Sales.Inventory I
JOIN Sales.TempData T ON I.ItemID = T.ItemID
AND I.RegionID = @RegionID;
```

The UPDATE statement locks the Sales.Inventory table when a single region is updated. You need to prevent the locking of the Sales.Inventory table when a single region is updated.
What should you do?

A. Modify the usp_Update stored procedure to use the NOLOCK table hint for the UPDATE statement.
B. Modify the usp_Update stored procedure to use the SERIALIZABLE table hint for the UPDATE statement.
C. Run the following Transact-SQL statement.
ALTER TABLE Sales.Inventory SET LOCK_ESCALATION = AUTO
D. Run the following Transact-SQL statement.
ALTER TABLE Sales.Inventory SET LOCK_ESCALATION = TABLE

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 175
You want to store amount of money with decimal precision 5 digits (like 10.35762). What column type sholud you use ?

A. Money format
B. Decimal format
C. ?
D. ?

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 176
You are a database developer on an instance of SQL Server 2008. You need to create a table to store information for construction projects undertaken by your company. You want to create a column to store a text narrative for each project. You want to minimize storage space, but allow the **Narrative** column to store up to

10000 bytes of data.
Which column definition should you use for the **Narrative** column?

A. `Narrative text`
B. `Narrative varbinary(max)`
C. `Narrative varchar(max)`
D. `Narrative varchar(10000)`

Answer: C
Section: (none)

Explanation/Reference:
This will create the **Narrative** column as a large-value type that can store up to 2 GB of data.
You should not define the **Narrative** column as a **text** data type because using the **text** data type should be avoided. A **text** data type can support 10000 bytes of data, but cannot be indexed. You should use the large-value type **varchar(max)** instead. You should not define the **Narrative** column as a **varbinary(max)** data type. A **varbinary** data type is used to store binary data, not text data. You should not define the **Narrative** column using `Narrative varchar(10000)`because this would generate an error. Even though using large-value types allows you to store more than 8060 bytes within a single row, each individual column can be no larger than 8000 bytes.

QUESTION 177
You are a database developer on an instance of SQL Server 2008. Your organization has a production database named **Prod** and a database
named **Dev** that is used by the development team. You want to create a CLR user-defined type (UDT) that will be passed as parameter input to a stored procedure. Using a .NET Framework language, you write and compile the necessary code to create an assembly.
In the **Dev** database, you register the assembly using a `CREATE ASSEMBLY` statement and create the type.
With no additional actions, which databases will be able to access the UDT?

A. neither the **Prod** nor **Dev** database
B. only the **Prod** database
C. only the **Dev** database
D. all databases on the SQL Server instance

Answer: C
Section: (none)

Explanation/Reference:
If you take no additional actions, only the **Dev** database may access the UDT you created. When you create a UDT in a database using the `CREATE ASSEMBLY` and `CREATE TYPE` statements, the scope of the UDT is restricted to that database. In this scenario, to access the UDT in the **Prod** database, you must execute the `CREATE ASSEMBLY` and `CREATE TYPE` statements on the **Prod** database.
All of the other options are incorrect because a UDT is only available to the database in which it was created.

QUESTION 178
You need to design a database solution that meets the following capabilities:
·Executes SQL Server Integration Services (SSIS) packages ·Executes Transact-SQL
·Schedules tasks
·Sends alerts
Which SQL Server component should you use?

A. Notification Services
B. Service Broker
C. SQL Mail
D. SQL Server Agent

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 179
You have a stored procedure that is used to set up maintenance tasks. The stored procedure executes every night.
The stored procedure contains three critical data manipulation language (DML) statements that operate against a single table.
You need to prevent users from modifying any data in the table while the stored procedure executes.
Which locking hint should you use?

A. NOLOCK
B. READCOMMITTED
C. REPEATABLEREAD
D. TABLOCKX

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 180
You are a database developer. You plan to design a database solution by using SQL Server 2008.
The database contains a table named Products.
The database has two stored procedures named ModifyProduct and RetrieveProducts. ModifyProduct updates a single row in the Products table. RetrieveProducts returns all rows from the Products table.
RetrieveProducts is used by a report. Users who run the report experience contention problems. You discover that RetrieveProducts is being blocked by ModifyProduct. The report must not include rows that are currently being modified. You need to ensure that the report is executed as quickly as possible. Which locking hint should you use in RetrieveProducts?

A. NOLOCK
B. NOWAIT
C. READPAST
D. READUNCOMMITTED

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 181
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. There're two stored procedures in the database. The two procedures are respectively named GetProduction and AlterProductions. GetProduction returns all rows from the Products table. AlterProductions updates a single row in the Products table. Contention problems occur when users run the report.
You check and find that Alterproduction blocks the GetProduction. Rows that are being modified can not be included in the report. You must make sure that the report is executed in a minimal time. In GetProduction, which locking hint should you use?

A. NOWAIT
B. NOLOCK
C. READPAST
D. READUNCOMMITTED

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 182
You are designing a database that contains a data definition language (DDL) trigger. The DDL trigger will provide the maximum amount of data available when any attempt is made to change the database schema. You need to design a table to meet the following requirements:
Accept the EVENTDATA information that is provided by the trigger Support the searching and retrieval of nodes and values Minimize development time
Which data type should you use?

A. nvarchar(max)
B. varchar(max)
C. varbinary
D. XML

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 183
Certkey.com - Make You Succeed To Pass IT Exams
Certkey 70-451

You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. Information on sales transactions of 300 chain stores is contained in the database. The solution is used by the marketing department to analyze daily sales patterns for each store. Users complain that the solution retrieves the required data slowly. So what should you do to make sure that the solution retrieves data as soon as possible?

A. On a view of the sales transactions, a nonclustered index is created

B. on a view, a clustered index should be created to aggregate the sales transactions
C. on a view, a nonclustered index should be created to aggregate the sales transactions
D. On a view, a covering index should be created to aggregate the sales transactions

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 184
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. There are three tables in the database. And the table below shows the structure of the three tables. The disk space usage should be reduced while the data types in the tables of the database cannot be altered. So what should you do to achieve this?

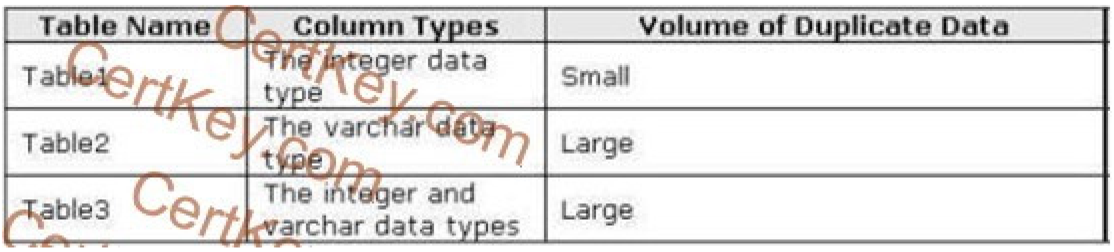


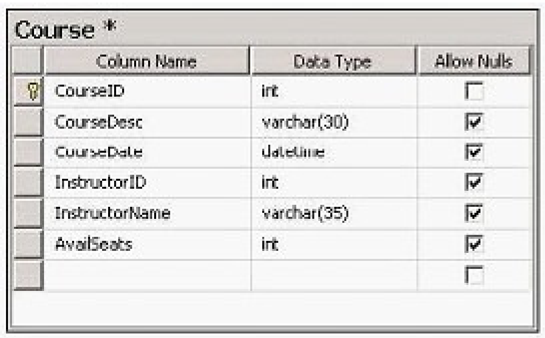
Table Name	Column Types	Volume of Duplicate Data
Table1	The integer data type	Small
Table2	The varchar data type	Large
Table3	The integer and varchar data types	Large

A. On all tables, implement row-level compression
B. On Table2, implement row-level compression; on Table1 and Table3, implement page-level compression.
C. On Table3, implement row-level compression; on Table1 and Table2, implement page-level compression.
D. On Table1, implement row-level compression; on Table2 and Table3, implement page-level compression

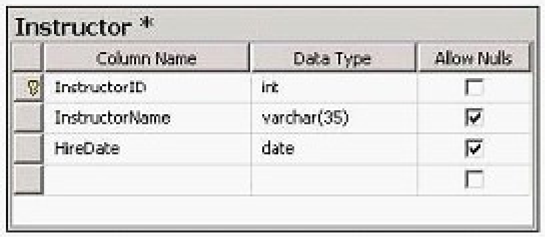
Answer: D
Section: (none)

Explanation/Reference:

QUESTION 185
You are the designer of a SQL database on an instance of SQL Server 2008. Your database contains a **Course** table and an **Instructor** table defined as follows:


Course *

	Column Name	Data Type	Allow Nulls
🔑	CourseID	int	☐
	CourseDesc	varchar(30)	✓
	CourseDate	datetime	✓
	InstructorID	int	✓
	InstructorName	varchar(35)	✓
	AvailSeats	int	✓
			☐


Instructor *

	Column Name	Data Type	Allow Nulls
🔑	InstructorID	int	☐
	InstructorName	varchar(35)	✓
	HireDate	date	✓
			☐

You want to create a `FOREIGN KEY` constraint to associate the two tables and prevent a user from deleting an instructor who is currently assigned to teach a course.
Which clause should you include in your `FOREIGN KEY` constraint?

A. `ON DELETE NO ACTION`
B. `ON DELETE CASCADE`
C. `ON DELETE SET NULL`
D. `ON DELETE SET DEFAULT`

Answer: A
Section: (none)

Explanation/Reference:
For example, in this scenario, you might use the following statement to create the necessary `FOREIGN KEY` constraint:

```
ALTER TABLE Course ADD CONSTRAINT FK_Instructor FOREIGN KEY (InstructorID)
REFERENCES Instructor (InstructorID)ON DELETE NO ACTION;
```

This statement includes the `ADD CONSTRAINT` clause of the `ALTER TABLE` statement to add a `FOREIGN KEY` constraint to the **Course** table. The `REFERENCES` clause specifies the referenced column in the parent table. When you include the `ON DELETE NO ACTION` clause, it ensures that referential integrity is maintained. If a referenced row in the parent table is deleted, then the delete fails if child rows exist. With the given statement, a user could not delete an instructor who was assigned to courses in the**Course** table. You should note that `NO ACTION` is the default for `ON DELETE` and `ON UPDATE`.
You should not include the `ON DELETE CASCADE` clause in your `FOREIGN KEY` constraint. If you did so and a user deleted an instructor that was assigned to teach a course, the delete operation would be performed, and all courses for the instructor would be deleted as well.
You should not include the `ON DELETE SET NULL` clause in your `FOREIGN KEY` constraint. If you did so and a user deleted an instructor who was assigned to teach a course, the**InstructorID** column would be set to `NULL` for all of the instructor's courses in the **Course** table. You should note that if the foreign key consists of multiple columns, all of the columns in the foreign key would be set to `NULL`.
You should not include the `ON DELETE SET DEFAULT` clause in your `FOREIGN KEY` constraint. If you did so and a user deleted an instructor who was assigned to teach a course, the**InstructorID** column would be set to the default value for all of that instructor's courses in the **Course** table. You should note that if the foreign key consists of multiple columns, all of the columns in the foreign key are set to the default value. In addition, you should ensure that each foreign key column has a `DEFAULT` definition; otherwise, the column will be assigned a `NULL` value.

QUESTION 186
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. There're two tables respectively named Storeroom and Commodity. On the StoreroomID column, there

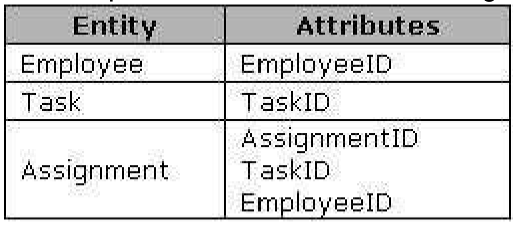
is a foreign key constraint between the Storeroom and Commodity tables. A row which has the StoreroomID value as 0 is contained in the Storeroom table. The 0 value shows that the Storeroom is deleted. The Storeroom records are deleted by certain transactions from the Storeroom table. You must make sure that the StoreroomID value in the Commodity table is set to 0 if a Storeroom is deleted. So what should you do?

A. On the Commodity table, a FOR DELETE trigger should be created to update the StoreroomID value to 0 in the Commodity table for the deleted Storeroom
B. On the Storeroom table, a FOR DELETE trigger should be created to update the StoreroomID value to 0 in the Commodity table for the deleted Storeroom
C. The ON DELETE property of the foreign key constraint should be set to NULL. Before this, on the StoreroomID column in the Commodity table, a default constraint should be created to set the value to 0.
D. The ON DELETE property of the foreign key constraint should be set to Default. Before this, on the StoreroomID column in the Commodity table, a default constraint should be created to set the value to 0

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 187
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. You are creating a database to support the office manager. The exhibit below shows the structure of the database. The database you design must meet the requirements below:
First, an employee can be given more than one task. Second, the task is deleted upon completion and when deleted, the associated assignment is deleted. Third, the employee link to the assignment is replaced with a NULL value when an employee is no longer available to complete a task. So what should you do to implement these requirements to maintain data integrity?



Entity	Attributes
Employee	EmployeeID
Task	TaskID
Assignment	AssignmentID TaskID EmployeeID

A. On the Employee, Task, and Assignment entities, DDL INSERT triggers should be created
B. On the TaskID and EmployeeID attributes in the Assignment entity, CHECK constraints should be created
C. On the TaskID and EmployeeID attributes in the Task and Employee entities, foreign Keys constraints should be created respectively. You specify the appropriate On Delete action after the Assignment entity is referenced.

D. On the TaskID and EmployeeID attributes in the Assignment entity, Foreign Keys constraints should be created. You specify the appropriate On Delete action. After the Task and Employee entities are referenced respectively.

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 188
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. Data from two storerooms are contained in the database in a table named Manufacures. The two storerooms are respectively called Storeroom1 and Storeroom2. And the database supports a warehousing application. A storeroom indicator field named storeroom_id is contained in the Manufacture table. Storeroom1 includes 275,000 items and Storeroom2 includes 55,000 items. A third-party application is used by the solutions. And the third-party application runs on SQL Server 2008 and uses a stored procedure that returns the storeroom inventory based on the warehouse_id parameter. You notice that when users query the storeroom1 inventory, users sometimes experience poor system performance. But the stored procedures cannot be modified in the application. So when the inventory of Storeroom1 is quried, what should you do to ensure the system performance?

A. On the storeroom_id column, a clustered index should be created
B. On the storeroom_id column, a non-clustered index should be created
C. A plan guide should be created that the MAXDOP query hint is set to 1
D. A plan guide that uses the OPTIMIZE FOR clause for Storeroom1 should be created.

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 189
You work in an International company named TAKEEEN. And you're in charge of the database of your company. There's a SQL Server 2008 database named TaskDB. A task management application which contains a module named Task is created to connect to the database. Users use a SQL Server login to log on to the application. The module assigns tasks to users. There is a Tasks table in the TaskDB database which stores the information about these tasks. Multiple columns are contained in the Tasks table, including the GettingDay and AboutTime columns.

Certkey.com - Make You Succeed To Pass IT Exams
Certkey 70-451

Look at the following requirements for the application:
Administrative users assigned to a database role named Task_Admin can update all task information in the Tasks table; Users assigned to a database role named UserA can update all task information columns except the GettingDay and the AboutTime columns in the Tasks table.
You have to design a strategy to meet the security requirements. So what should you do? (choose more than one)

A. On the Tasks table, Update permissions should be assigned to the Task_Admin role
B. The Task_Admin role should be added to the db_accessadmin fixed database role

C. On the Tasks table, Update permissions should be assigned to the UserA role for each column except the GettingDay and AboutTime columns
D. You should prevent the User1 role from updating the GettingDate and AboutTime columns by using the Is_Member function. Before you perform this, On the Tasks table, an INSTEAD OF trigger should be created.

Answer: AC
Section: (none)

Explanation/Reference:

QUESTION 190
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. The database application has a table named Deals. Millions of rows are contained in the table. The table has multiple columns. The columns include deal_id and deal_date. On the deal_date column, there is a nonclustered index. On the deal_id column, there is a clustered index. Look at the the following query:
SELECT transaction_id, transaction_date, transaction_notes FROM transactions
WHERE transaction_type_id = 'FXO'
AND transaction_date between @start_date and @end_date You notice that the query above execute slowly. What should you do to make sure that the query retrieves data by using as little time as possible?

A. On the transaction_type_id column, a nonclustered index should be created
B. On the transaction_date and transaction_type_id columns, a nonclustered index should be created.
C. On the transaction_date and transaction_type_id columns, a nonclustered index should be created, and you should include the transaction_notes column.
D. On the transaction_date column, a nonclustered index should be created and the transaction_type_id and transaction_notes columns should be included.

Answer: C
Section: (none)

Explanation/Reference:

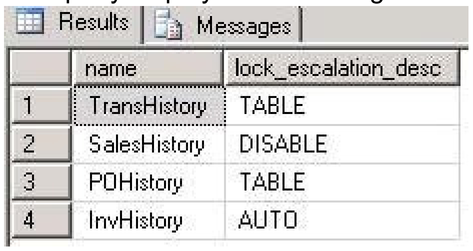
QUESTION 191
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. There's a database contains a view. This view contains a WHERE clause that filters specific records and allows data updates. You have to prevent data modifications which do not conform to the WHERE clause while using as little effort as possible. So what should you do?

A. On the view, an INSTEAD OF trigger should be created
B. On the view, a unique clustered index should be created
C. You should add the WITH CHECK OPTION clause to alter the view.
D. You should add the WITH SCHEMABINDING clause to alter the view

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 192
You manage a database on an instance of SQL Server 2008. From SQL Server Management Studio, you issue the following query:
```
SELECT name, lock_escalation_desc FROM sys.tablesWHERE name LIKE '%History';
```
The query displays the following results:



	name	lock_escalation_desc
1	TransHistory	TABLE
2	SalesHistory	DISABLE
3	POHistory	TABLE
4	InvHistory	AUTO

All four history tables are partitioned tables.
With the displayed settings, for which table(s) will lock escalation occur to the partition level?

A. only for the **SalesHistory** table
B. only for the **InvHistory** table
C. for the **TransHistory** and **POHistory** tables
D. for the **TransHistory**, **POHistory**, and **InvHistory** tables

Answer: B
Section: (none)

Explanation/Reference:
In this scenario, lock escalation will occur to the partition level only for the **InvHistory** table. By default, SQL Server will escalate locks to the table level when required. When an excessive number of row-level locks are encountered, SQL Server will escalate the lock to a higher level. This causes the entire table to be locked, which is usually sufficient and reduces memory consumption. However, with partitioned tables, you may not want to lock the entire table, but only a single partition, such as the partition containing rows being modified. This would allow queries against other partitions in the table to execute successfully without being blocked.
You can use the `LOCK_ESCALATION` setting for a table to control how SQL Server will escalate locks.
The `LOCK_ESCALATION` setting can have one of the following values:
`TABLE`: Lock escalation is performed to the table level, which is the default.
`AUTO`: Lock escalation is performed to the table level for non-partitioned tables, and to the partition level for partitioned tables.
`DISABLE`: Lock escalation is disabled, and lock escalation does not typically occur. SQL Server only escalates locks in specific situations where it is absolutely required to ensure data integrity.

In this scenario, the **InvHistory** table has a `LOCK_ESCALATION` setting of `AUTO`. Therefore, if you issue an update against a single partition in the table that causes lock escalation, locks will be escalated to the partition level. This will allow users to query other partitions of the **InvHistory** table as needed. You can use the `ALTER TABLE` statement to set the lock escalation for a table. For example, the following statement would set the `LOCK_ESCALATION` setting for the **POHistory** table to `AUTO`:
```
ALTER TABLE POHistory SET (LOCK_ESCALATION=AUTO);
```
All of the other options are incorrect. The **TransHistory** and **POHistory** tables have a `LOCK_ESCALATION` setting of `TABLE`, which is the default setting. With this setting, SQL Server will escalate locks to the table level, rather than the partition level. The **SalesHistory** table has a `LOCK_ESCALATION` setting of `DISABLE`, which indicates that lock escalation will not typically occur.

QUESTION 193

You are a database developer on an instance of SQL Server 2008. Your **HR** database contains a **Course** table that contains descriptions for
training courses provided by your company.
The **HR** database has been enabled for full-text searches. You have created a full-text index on the **Description** column in the **Course** table.
You execute the following Transact-SQL:

```
SELECT Description
FROM HR.dbo.Course
WHERE CONTAINS (Description, 'FORMSOF (INFLECTIONAL, budget)');
```

What will be returned?

A. only the **Description** values for rows in the **Course** table that contain a form of the word budget
B. the **Description** values for rows in the **Course** table that contain a form of the word budget along with a relevance ranking
C. only the **Description** values for rows in the **Course** table that contain a word with a meaning similar to the word budget
D. only the **Description** values for rows in the **Course** table that contain the word budget

Answer: A
Section: (none)

Explanation/Reference:
The given statement will return only the **Description** values for rows in the **Course** table that contain a form of the word budget. The `CONTAINS`
predicate can be used in a `WHERE` clause to search character-based columns for an inflectional form of a specific word, a specific word or phrase, a
word or phrase that is near another word or phrase, or words or phrases with weighted values. In this scenario, you used `WHERE CONTAINS`
`(Description, 'FORMSOF (INFLECTIONAL, budget)')`. This would return all **Description** values that contained a form of the word budget.
For example, SQL Server would return rows with a **Description** column containing the words budget, budgeted, budgeting, budgets, or budget's.
The given statement will not return only the **Description** values for rows in the **Course** table that contain the word budget. The `CONTAINS` predicate
can be used to perform such a search. The following statement would accomplish this:

```
SELECT Description
FROM HR.dbo.Course
WHERE CONTAINS (Description, 'budget');
```

You can use the `CONTAINS` predicate to find words with similar meanings. To accomplish this, you must first configure the appropriate
thesaurus file to include the desired synonyms. Then, you could use the following statement:

```
SELECT Description
FROM HR.dbo.Course
WHERE CONTAINS (Description, 'FORMSOF (THESAURUS, budget)');
```

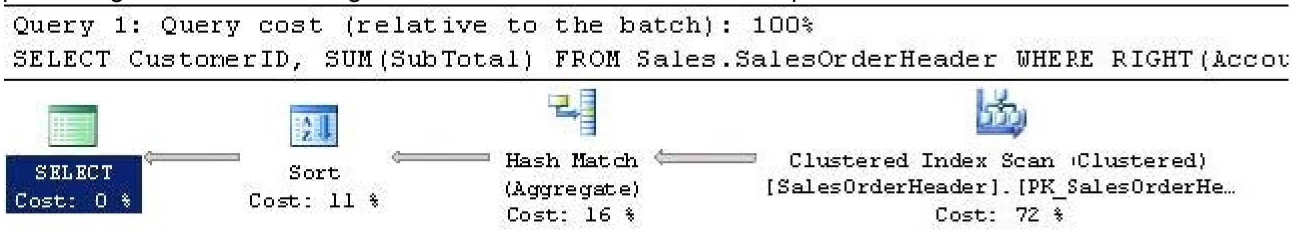
You can also use the `FREETEXT` predicate to search for matches with similar meaning rather than the exact wording. With the `FREETEXT` predicate,
strings are first separated into words, word-stemming is performed to determine other word forms, and then the result is passed through the
thesaurus to identify words with similar meanings. The primary difference between using the thesaurus with the `CONTAINS` predicate and using the
`FREETEXT` predicate is that `CONTAINS` only uses the thesaurus and does not perform word stemming. The `FREETEXT` predicate accepts two
parameters. The first parameter specifies one or more full-text enabled columns to search. An asterisk (`*`) in the first parameter indicates that all fulltext
enabled columns should be searched. The second parameter is a character string that can contain words or phrases. If the search should be

performed on phrases, the search string should be enclosed in double quotation marks instead of single quotation marks. The following statement
would search the **Description** column of the **Course** table for descriptions associated with budgeting tasks:

```
SELECT Description
FROM HR.dbo.Course
WHERE FREETEXT (Description, "budgeting tasks");
```

The given statement will not return the **Description** values for rows in the **Course** table that contain a form of the word budget along with a
relevance ranking because the `CONTAINS` predicate does not return a relevance ranking. You can use the `CONTAINSTABLE` function in the `FROM`
clause of a `SELECT` statement to perform a similar search and return a relevance ranking. The `CONTAINSTABLE` function returns a table containing
rows matching the specified selection criteria. Each row of the table contains a **Key** column and a **Rank** column. The **Key** column contains the fulltext
key, and the **Rank** column contains a relevance ranking.

QUESTION 194
You are a database developer on an instance of SQL Server 2008. You have a large production database that stores sales information for your company.
You have a query that summarizes data in the **SalesOrderHeader** table. You view its graphical execution plan using SQL Server Management Studio. The actual execution plan is as follows:



```
Query 1: Query cost (relative to the batch): 100%
SELECT CustomerID, SUM(SubTotal) FROM Sales.SalesOrderHeader WHERE RIGHT(Accou
```

Depending on the number of rows in the table, which operations in the execution plan might cause the **tempdb** database to grow significantly?

A. only the **Sort** operation
B. only the **Hash Match** operation
C. both the **Sort** and **Hash Match** operations
D. only the **Clustered Index Scan** operation

Answer: C
Section: (none)

Explanation/Reference:
The **Sort** and **Hash Match** operations might cause the **tempdb** database to grow significantly. The **tempdb** database stores temporary data, including internal objects created by SQL Server. Intermediate sort results and intermediate results of hash joins and hash aggregates are also stored in**tempdb**. Therefore, the **Sort** operation and the **Hash Match** operation in the execution plan might cause the **tempdb** database to grow significantly, depending on the number of rows in the underlying table.
All of the other options are incorrect. A **Clustered Index Scan** operation occurs when SQL Server scans a clustered index or a range in a clustered index, rather than accessing a specific index entry. A **Clustered Index Seek** accesses a specific index entry and provides better performance than a **Clustered Index Scan**. A **Clustered Index Scan** would not be likely to cause the size of the **tempdb** database to increase significantly.

QUESTION 195

You are a database developer on an instance of SQL Server 2008. You are creating an application that will be used by the inventory department. You write the following Transact-SQL code:

```
BEGIN TRANSACTION
 INSERT INTO InvLookup VALUES (1, 'Finished Goods');

 BEGIN TRANSACTION
  INSERT INTO InvLookup VALUES (7, 'Component');
 COMMIT TRANSACTION;

 UPDATE InvTemp SET ProcFlg = 'X'WHERE InvID IN (SELECT InvID FROM InvMaster);
 UPDATE InvMaster SET ProcFlg = 'Y';

 ROLLBACK TRANSACTION;
```

Assuming all DML statements in both the inner and outer transaction process successfully, which statements will be committed?

A. only the statements in the inner transaction
B. only the statements in the outer transaction
C. none of the statements in either the inner or outer transaction
D. all of the statements in both the inner and outer transactions

Answer: C
Section: (none)

Explanation/Reference:
When using a nested transaction, the outer transaction controls which modifications are committed or rolled back. With the given Transact-SQL, you issue an `INSERT` statement within the outer transaction. Then, you begin another transaction using the `BEGIN TRANSACTION` statement and issue an `INSERT` statement and a `COMMIT TRANSACTION` statement within this inner transaction. With a non-nested transaction, the `COMMIT TRANSACTION` statement would commit the changes. However, with nested transactions, any `COMMIT TRANSACTION` or `ROLLBACK TRANSACTION` statement is ignored, and the outer transaction controls which modifications are committed or rolled back. In this scenario, you issue a `ROLLBACK TRANSACTION` statement at the end of the outer transaction that rolls back all changes made by both transactions. If you had issued a `COMMIT TRANSACTION` statement for the outer transaction, then all of the DML statements in both transactions would have been committed.
All of the other options are incorrect because when you use nested transactions, the outer transaction controls which modifications are committed or rolled back.

QUESTION 196
You are a database developer on an instance of SQL Server 2008. You have a complex query that is performing poorly. You examine the query's execution plan and determine that the query optimizer has not chosen an index that will provide the best performance. Which optimizer hint can you include for the query to force the query to use a specific index rather than the index chosen by the query optimizer?

A. Optimize for
B. table hint
C. fast
D. force order

Answer: B
Section: (none)

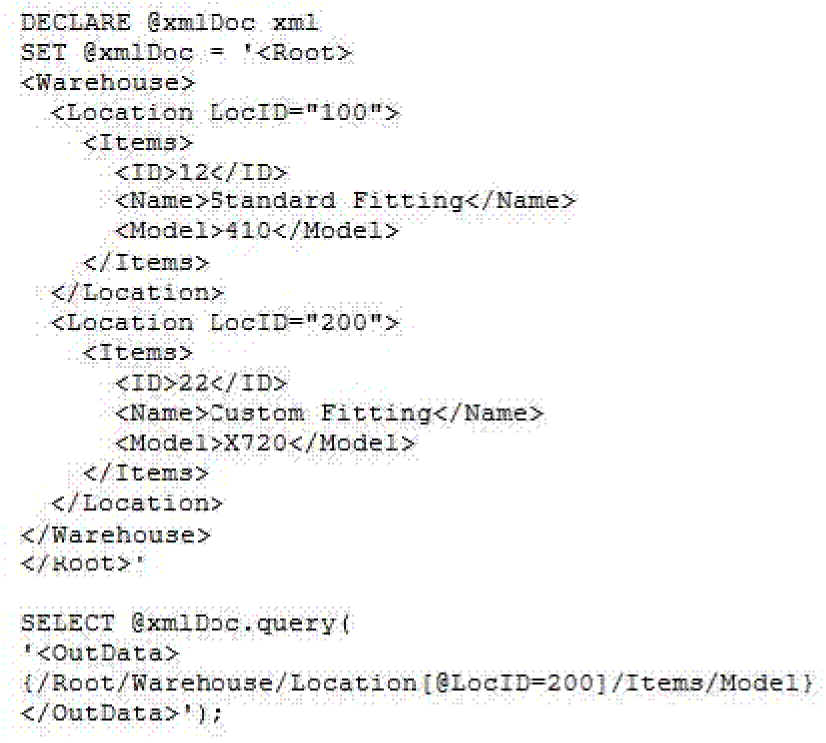
Explanation/Reference:

You can use the `TABLE HINT` query hint. In this scenario, you want to force a query to use a specific index. You can do so using an `INDEX` table hint specified using the `TABLE HINT` query hint.
You should not use the `OPTIMIZE FOR` query hint because it is used to optimize queries based on a specific value for a local variable used in the query.
You should not use the `FAST` query hint. The `FAST` query hint optimizes the query to quickly retrieve a specified number of rows. You would use the `FAST` query hint to quickly return a fewer number of rows from a large table. For example, you might use the following query to quickly retrieve the first 200 rows from the **Products** table:

```
SELECT * FROM ProductHistoryORDER BY ProdStyle DESCOPTION (FAST 200);
```

When this query executes, the first 200 rows in the result set will be returned as quickly as possible, and then the query will return the remainder of the result set when the query completes.
You should not use the `FORCE ORDER` query hint. The `FORCE ORDER` query hint controls how the query's join order is handled when a query is being optimized. Specifying `FORCE ORDER` indicates that query optimization will not affect the join order specified in the query.

QUESTION 197
You are a database developer on an instance of SQL Server 2008. You execute the Transact-SQL code shown in the exhibit. (Click the **Exhibit(s)** button.)
Which XML will be extracted?

Exhibit:


```
DECLARE @xmlDoc xml
SET @xmlDoc = '<Root>
<Warehouse>
  <Location LocID="100">
    <Items>
      <ID>12</ID>
      <Name>Standard Fitting</Name>
      <Model>410</Model>
    </Items>
  </Location>
  <Location LocID="200">
    <Items>
      <ID>22</ID>
      <Name>Custom Fitting</Name>
      <Model>X720</Model>
    </Items>
  </Location>
</Warehouse>
</Root>'

SELECT @xmlDoc.query(
'<OutData>
{/Root/Warehouse/Location[@LocID=200]/Items/Model}
</OutData>');
```

A. `<OutData>`
`<Items> <ID>22</ID><Name>Custom Fitting</Name><Model>X720</Model></Items></OutData>`

B. `<OutData>`
`<Model>410</Model><Model>X720</Model></OutData>`

C. `<OutData><Model>X720</Model></OutData>`

D. `<OutData>`
`<Location LocID="200"> <Items> <ID>22</ID><Name>Custom Fitting</Name><Model>X720</Model></Items></Location></OutData>`

Answer: C
Section: (none)

Explanation/Reference:
The **query()** method is used to query and retrieve XML elements and attributes from an XML instance. The method accepts a string argument to specify the element or attribute that should be extracted. In this scenario, you specify a query string of `{/Root/Warehouse/Location[@LocID=200]/Items/Model}`. This extracts the `<Model>` element, which is a child of the `<Items>` element, from within the `<Location>` element with a **LocID** value of 200.
The given code will not extract the XML that contains two `<Model>` elements within an `<OutData>` element. This result could be generated from `@xmlDoc` using the following `SELECT` statement:

```
SELECT @xmlDoc.query('<OutData> {/Root/Warehouse/Location/Items/Model}</
OutData>');
```

The given code will not extract the XML that contains an `<Items>` element including the `<ID>`, `<Name>`, and `<Model>` subelements. This result could be generated from `@xmlDoc` using the following `SELECT` statement:

```
SELECT @xmlDoc.query('<OutData> {/Root/Warehouse/Location[@LocID=200]/Items}</
OutData>');
```

The given code will not extract the XML that contains a `<Location>` element including subelements. This result could be generated from `@xmlDoc`using the following `SELECT` statement:

```
SELECT @xmlDoc.query('<OutData> {/Root/Warehouse/Location[@LocID=200]}</
OutData>');
```

QUESTION 198
You have to protect yourself against SQL injections what should you do:

A. Parse all input for: -- /* */ ; '
B. Concatenate user input that is not validated.
C. Accept the following strings in fields from which file names can be constructed: AUX, CLOCK$, COM1 through COM8, CON, CONFIG$, LPT1 through LPT8, NUL, and PRN.
D. Use stored procedures with unfiltered input.

Answer: A
Section: (none)

Explanation/Reference:
http://msdn.microsoft.com/en-us/library/ms161953.aspx

When you can, reject input that contains the following characters.

Input character

Meaning in Transact-SQL

;

Query delimiter.

'

Character data string delimiter.

--

Comment delimiter.

/* ... */

Comment delimiters. Text between /* and */ is not evaluated by the server.

xp_

Used at the start of the name of catalog-extended stored procedures, such as xp_cmdshell.

QUESTION 199
You maintain SQL Server 2008 instances for a large global manufacturer. You currently have two SQL Server 2008 instances, **SQL01** and **SQL02**.
SQL01 contains several large production databases that are directly accessed by users, and also contains numerous third-party applications.
The **Manufacturing** database is used 24x7 to support all activities related to the manufacturing process. The database uses the full recovery
model.
You want to provide failover capability for the **Manufacturing** database. You also need to provide read-only reporting access to manufacturing
details. The data is used for reports and does not have to be the most recent data, but it must be readily available and current.
You want to accomplish this with the least effort possible and allow for minimal ongoing administration effort.
Which action should you take?

A. Periodically use the Import and Export Wizard in BIDS to transfer the needed data from **SQL01** to **SQL02** to be used for reporting.
B. Implement a partitioned view of the data and provide query access to the view.
C. Implement database mirroring with **SQL01** as the principal server and **SQL02** as the mirror server. Periodically create a snapshot of the mirror
database for reporting.
D. Configure log shipping.

Answer: C
Section: (none)

Explanation/Reference:
You should implement database mirroring with **SQL01** as the principal server and **SQL02** as the mirror server, and periodically create a snapshot of
the mirror database for reporting. Database mirroring provides failover capability by maintaining two separate database copies on different SQL
Server instances. The principal server contains the currently used data, and the mirror server maintains a copy of the data that can be used if the
principal server fails. To implement mirroring, you back up the database on the principal and restore it to the mirror server with no recovery.
Although clients cannot directly access the mirror database, you can create a database snapshot on the mirror database and provide read-only
access on the snapshot. Clients would be able to access the data as it existed when the snapshot was taken to produce reports. This solution would
meet the failover and reporting requirements in this scenario.

You should not configure log shipping. Log shipping would be more difficult to administer than using database mirroring with a snapshot on the
mirror. With log shipping, transaction log backups from a primary database are automatically transferred to another database and then applied.
You should not implement a partitioned view of the data and provide query access to the view because this would not meet the failover
requirements in this scenario. Partitioned views are used when you have similar data stored in multiple tables and want to create a view to allow
access to all of the data as if it were stored in a single table. Partitioned views are implemented by creating a view that queries several tables and
combines the results using the `UNION ALL` operator. A partitioned view can be implemented across multiple servers to improve performance and
increase data availability, and to make a database solution more scalable. This decentralizes the processing load and administration effort to
individual servers, but the view provides for collective data access.
You should not periodically use the Import and Export Wizard in BIDS to transfer the needed data from **SQL01** to **SQL02** to be used for reporting.
Although you could transfer information to another server instance and use the data for reporting purposes, this would not provide the failover
capability required in this scenario. The Import and Export Wizard creates SSIS packages to perform the data transfer tasks. These packages can
be saved and reused.

QUESTION 200
You have a table named Sales that has the following three columns: Duty, Freight, and Customs.

You need to add new column to table which would show Sales+Freight+Customs.

Which option should you chose to implement?

A. Persisted Computed Column
B. Computed Column
C. Numeric data type
D. Money data type

Answer: C
Section: (none)

Explanation/Reference:
Unless otherwise specified, computed columns are virtual columns that are not physically stored in the table. Their values are recalculated every time they are referenced in a query. The Database Engine uses the PERSISTED keyword in the CREATE TABLE and ALTER TABLE statements to physically store computed columns in the table. Their values are updated when any columns that are part of their calculation change.

QUESTION 201
You are a database developer on an instance of SQL Server 2008 for a large retail sales organization. Your **SalesOrderDetail** table contains millions of rows with details of all sales orders.
Each day, you need to run a query that joins the **SalesOrderDetail** table with the **Product** table, performs a complex aggregation for each row of **SalesOrderDetail**, and writes the results to the **SalesStats** table.
You want to ensure the most current data is used in the aggregations, but that the query does not adversely affect sales representatives that are taking orders from customers.
What should you do?

A. Process the query within a transaction and set the transaction's isolation level to `REPEATABLE READ`.

B. Create a stored procedure that implements transactions to process rows in the **SalesOrderDetail** table in small batches.
C. Create a query that builds a temporary table and run your query against the temporary table.
D. Create a CLR user-defined aggregate.

Answer: B
Section: (none)

Explanation/Reference:
. Using a transaction, you would be able to read the most recent data to perform your aggregations. However, if you initiated the transaction and performed all the aggregations at one time, other users' access to the table would be adversely affected. Splitting the work into small batches would allow you to perform periodic commits and not lock the table for the entire duration of the query.
You should not process the query within a transaction and set the transaction's isolation level to `REPEATABLE READ`. The `REPEATABLE READ` isolation level prevents statements from reading data that has been modified by other transactions but not yet committed. It also prevents other transactions from modifying data that has been read by the current transaction until the current transaction completes. This would adversely affect users taking orders because they could not modify any data that was read by your transaction until your transaction completed.
You should not create a query that builds a temporary table and run your query against the temporary table because this approach would not allow you to perform aggregations on the most up-to-date data.
You should not create a CLR user-defined aggregate. CLR user-defined aggregates are created in a .NET Framework language. They allow you to create custom aggregate functions that users can call like other built-in aggregate functions, such as `SUM`, `MAX`, or `COUNT`. Using a CLR user-defined aggregate would not ensure that other users were not affected when your query executed.

QUESTION 202
You are a database developer for at large mail processing facility. You have a **Prod** database that contains tables used by many in-house
applications, as well as third-party applications.
You have an XML document that contains the following:
```
<Root>
<Station ID="12" Type="Collation">
<PieceCount>5512</PieceCount>
<CustomerID>30</CustomerID>
</Station>
<Station ID="25" Type="Postal Sort">
<PieceCount>2503</PieceCount>
<CustomerID>30</CustomerID>
</Station>
<Station ID="14" Type="Folder">
<PieceCount>10548</PieceCount>
<CustomerID>65</CustomerID>
</Station>
<Station ID="12" Type="Insertion">
<PieceCount>27690</PieceCount>
<CustomerID>49</CustomerID>
</Station>
</Root>
```
You are creating a query to retrieve data from the XML and insert it into a temporary table.
Which XML method should you use in the `SELECT` list of your query?

A. query()
B. value()
C. nodes()

D. modify()

Answer: B
Section: (none)

Explanation/Reference:
You should use the **value()** method in the `SELECT` list of your query. The **value()** method accepts two arguments. The first argument is a string
XQuery expression, and the second argument is a string containing a SQL Server data type. The **value()** method extracts a single value using the
XPath expression and returns it as the specified type. For example, in this scenario, you could load the XML into an **xml** variable named `@x` and use
the following statement to insert rows into a temporary table:

```
SELECT
t.c.value('@ID[1]', 'int') AS StationID,
t.c.value('@Type[1]', 'varchar(35)') AS ProcessType,
t.c.value('PieceCount[1]', 'int') AS NumPieces
INTO #PieceCounts
FROM @x.nodes('//Root/Station') AS t(c);
```

With this statement, the **value()** method extracts values of the `ID` and `Type` attributes of the `<Station>` element and the value of the
`<PieceCount>` element. Each element or attribute specified is returned with the specified data type, and the `INTO` clause inserts the result into a
temporary table named **#PieceCounts**. The `[1]` at the end of each string expression indicates that the information is extracted from a single
element. The **value()** method must return a single value. However, in this scenario, the XML contains multiple `<Station>` elements. Therefore, the
nodes() method must be used in the `FROM` clause. With the given statement, the following data would be inserted into the **#PieceCounts** table.
You should not use the **query()** method. The **query()** method is used to query and retrieve XML elements and attributes from an XML instance as
untyped XML, not a relational result set. The method accepts a string XQuery expression that determines which elements and element attributes
are extracted.
You should not use the **nodes()** method. The **nodes()** method accepts an XQuery string and returns all of the specified nodes as a result set. You
can use the **nodes()** method in the `FROM` clause of a query to process multiple XML nodes using the **value()** method in the `SELECT` list.
You should not use the **modify()** method. The **modify()** method is used to change the value of an **xml** type variable or column. The **modify()**
method can only be invoked using the `SET` clause of an `UPDATE` statement. The method accepts a string argument containing an XML Data
Manipulation Language (DML) statement, such as **replace value of**, **insert**, or **delete**. The statement is then used to update the corresponding
element or attribute in the XML.

QUESTION 203
You are a database developer on an instance of SQL Server 2008. Your **Sales** database contains current and historical sales information for your company.
The development team has recently created a user-defined table-valued function in .NET that accepts some input parameters and returns a list of customers ranked from highest to lowest based on their sales volume.
You have registered the assembly and issued the following statement:

```
CREATE FUNCTION dbo.GetRanked(@p1 int, @p2 char(5), @p varchar(20))RETURNS TABLE
(CustID int,CustName varchar(35),Volume money,Ranking int) WITH EXECUTE AS
CALLERAS EXTERNAL NAME MyAssembly.CLRFunctions.GetRankedList;
```

You use the result set returned by the function in many queries that include different values with the `TOP` keyword, and you want the queries to execute as quickly as possible.

Which action should you take?

A. Re-create the **GetRanked** function to include an `ORDER` clause.
B. Include the `FAST` query hint in each of your queries.
C. Re-create the **GetRanked** function to use `EXECUTE AS SELF`.
D. Create a Transact-SQL stored procedure that accepts a table-valued input parameter and inserts the values into another table, and create an index on the **Ranking** column of the table

Answer: A
Section: (none)

Explanation/Reference:
In this scenario, the CLR user-defined function always returns the result set in a specified order. You can optimize performance of your queries by specifying the `ORDER` clause for your CLR function. This will prevent the need to sort the result each time you run a query that includes the `TOP` keyword, and also will provide optimum performance. In this scenario, you could drop the **GetRanked** function and re-create it using the following statement:

```
CREATE FUNCTION dbo.GetRanked(@p1 int, @p2 char(5), @p varchar(20))RETURNS TABLE
(CustID int,CustName varchar(35),Volume money,Ranking int) WITH EXECUTE AS
CALLERORDER(Ranking)AS EXTERNAL NAME MyAssembly.CLRFunctions.GetRankedList;
```

You should not create a Transact-SQL stored procedure that accepts a table-valued input parameter and inserts the values into another table, and create an index on the **Ranking** column of the table. In this scenario, there is no need to code an additional stored procedure, nor require additional storage. Therefore, using an ordered table-valued function would be a better choice.
You should not include the `FAST` query hint in each of your queries. The `FAST` query hint is used to quickly return a certain number of rows from a query, and then return the remainder of the rows after the query completes. You would use the `FAST` query hint if you had a long-running query and
you wanted to return the initial portion of the result while the query continued execution.
You should not re-create the **GetRanked** function to use `EXECUTE AS SELF`. The `EXECUTE AS` clause is used to control the security context under which the function executes. This would have no impact on query performance, but rather would determine the permissions required to use the function.

QUESTION 204
You are a database developer on an instance of SQL Server 2008. You maintain a large **Sales** database for a retail distributor.
You currently have a stored procedure that is used by several applications, including your online order entry application. The stored procedure accepts an input parameter, `@territory`, and returns information about sales orders for the specified territory using a single `SELECT` statement. Then, the stored procedure executes several other SQL statements to gather details and perform DML operations for each sales order using the **SalesOrderID** returned by the first query.
The stored procedure's performance is unacceptable. After analyzing the stored procedure and the statements it executes, you want to ensure that a new execution plan is generated for the initial territory query each time the stored procedure is called.
Which action should you take?

A. Re-create the stored procedure including the `WITH RECOMPILE` clause.
B. Call the **sp_create_plan_guide** system stored procedure to create a plan guide for the initial query that includes the `RECOMPILE` query hint.
C. Set the `PARAMETERIZATION` option to `FORCED`.
D. Recompile the stored procedure using the **sp_recompile** system stored procedure.

Answer: B
Section: (none)

Explanation/Reference:
Plan guides can be used to optimize queries without modifying the statement directly. In a plan guide, you include the query to be optimized, and either an `OPTION` clause with query hints, or a query plan that should be used during optimization. The **sp_create_plan_guide** system stored procedure accepts the following parameters:
`@name` - Specifies a unique plan guide name.
`@stmt` - Specifies the Transact-SQL statement or batch associated with the plan guide.
`@type` - Specifies the type of plan guide. Valid values are `OBJECT`, `SQL`, and `TEMPLATE`.
`@module_or_batch` - Specifies the module name if the plan guide applies to a specific module, such as a stored procedure.
`@params` - Specifies an optional list of parameters to be used for SQL and template plan guide types.
`@hints` -Specifies an `OPTION` clause that includes hints to be used during optimization.

After you create a plan guide for a SQL statement, SQL Server matches the Transact-SQL statement to the plan guide when the query executes, and uses the specified `OPTION` clause hints or query plan for the statement. In this scenario, you wanted to force recompilation of the first `SELECT` statement in the stored procedure. You can accomplish this by creating a plan guide for the statement that includes the `RECOMPILE` query hint. This will force statement-level recompilation, and recompile the statement each time the stored procedure executes.
Plan guides can be especially useful if you need to add hints to the query to optimize the query without directly modifying it, such for deployed applications for which you cannot or do not want to modify existing code.
You should not re-create the stored procedure including the `WITH RECOMPILE` clause. Including the `WITH RECOMPILE` clause would force recompilation for the entire stored procedure each time it executed. In this scenario, you only wanted to force recompilation for the initial `SELECT` statement.
You should not set the `PARAMETERIZATION` option to `FORCED`. Forced parameterization can limit the frequency of query compilations and recompilations in a database. This option can only be applied to the entire database, not used to recompile a single query within a stored procedure.
You should not recompile the stored procedure using the **sp_recompile** system stored procedure because this would recompile the entire stored procedure the next time it executed. In this scenario, you only needed to recompile the first query. In addition, you wanted recompilation for the first query to occur each time the stored procedure executed.

QUESTION 205
You are a database administrator on an instance of SQL Server 2008. The development team has recently created some user-defined functions in .NET that you would like to access from SQL Server.
One of the functions, named **GetSample**, accepts a single input parameter and returns a result set containing a random sampling of data from several tables in your current database. Users need to be able to issue queries that use the generated sample.
You successfully execute the following Transact-SQL statements:
```
CREATE ASSEMBLY CLRAssemblyFROM 'C:\CLRTest.dll' WITH PERMISSION_SET = SAFE;GO
CREATE FUNCTION MyFunction (@parm int)RETURNS varchar AS EXTERNAL NAME
AssemblyName.NamespaceName.ClassName.GetSample;
```
When you attempt to access the function, an error is returned.
Which action should you take?

A. Re-register the assembly with the `EXTERNAL_ACCESS` permission set and re-create the function as an inline table-valued function.
B. Remove the `EXTERNAL NAME` clause from the `CREATE FUNCTION` statement and drop and re-create the function.
C. Drop the function and re-create it to return a **table**.
D. Re-create the desired functionality using a Transact-SQL stored procedure.

Answer: C
Section: (none)

Explanation/Reference:
In this scenario, the .NET UDF returns a result set, not a scalar value. When you issued the `CREATE FUNCTION` statement, you specified that the function returned a scalar value. You need to re-create the function with a `RETURNS` clause that specifies a tabular result set instead of a scalar value. You would specify the details of the columns returned by the **GetSample** CLR UDF.
You should not re-register the assembly with the `EXTERNAL_ACCESS` permission set and re-create the function as an inline table-valued function. In this scenario, there is no need for the assembly to be given the `EXTERNAL_ACCESS` permission set because it uses tables within your current database. The `EXTERNAL_ACCESS` permission set allows access to the registry, the file system, environment variables, and unmanaged code. In addition, you would not re-create the function as an inline table-valued function. An inline table-valued function would not be able to provide the details of the table returned by the **GetSample** function. Inline table-valued functions are defined by including a single `SELECT` statement without specifically naming the columns of the result. For example, the following statement would create an inline-table valued function without specifying the details of the returned **table** value.
```
CREATE FUNCTION dbo.GetData (@parm int)RETURNS table AS RETURN (SELECT * FROM
MyTableWHERE MyColumn = @parm);
```
You should not remove the `EXTERNAL NAME` clause from the `CREATE FUNCTION` statement and drop and re-create the function. The `EXTERNAL NAME` clause must be specified when creating a CLR UDF to identify the .NET function that will be referenced.
You should not re-create the desired functionality using a Transact-SQL stored procedure. In this scenario, you can access the previously created CLR function without additional development effort.

QUESTION 206
You are a database developer on an instance of SQL Server 2008. You are creating an application that will use transactions. Your transaction processing strategy should not allow any concurrency side effects, such as dirty reads, phantom reads, or repeatable reads. You want to use the isolation level that will meet these requirements, but consume the fewest system resources. Which transaction isolation level should you use?

A. `READ COMMITTED`
B. `REPEATABLE READ`
C. `SERIALIZABLE`
D. `SNAPSHOT`

Answer: D
Section: (none)

Explanation/Reference:
Isolation levels determine the degree of isolation for a transaction from changes made to resources or data by other transactions. The `SNAPSHOT` isolation level ensures that the data read by any statement in a transaction will remain consistent until the transaction is complete. This level does not allow dirty reads, nonrepeatable reads, or phantom reads. To set the isolation level to `SNAPSHOT`, the `ALLOW_SNAPSHOT_ISOLATION` database option, which is `OFF` by default, must be set to `ON`. To define an isolation level for transactions, you can use the `SET TRANSACTION ISOLATION LEVEL` statement.
You should not use the `SERIALIZABLE` isolation level. The `SERIALIZABLE` isolation level is the most restrictive isolation level in which the transactions are completely isolated from one another. The `SERIALIZABLE` isolation level also prevents currency side effects, but the `SNAPSHOT` isolation level is less restrictive because it uses row versions from when the transaction started and only acquires table locks, not page or row locks. Although the `SERIALIZABLE` isolation level does meet the requirements, it uses more system resources because it is a higher isolation level.
You should not use the `READ COMMITTED` isolation level. The `READ COMMITTED` isolation level, which is

the default isolation level, prevents statements from reading data that has been modified by other transactions but not committed. This isolation level does not allow dirty reads, but does allow phantom reads and nonrepeatable reads.
You should not use the `REPEATABLE READ` isolation level. This isolation level does not allow dirty reads or nonrepeatable reads, but does allow phantom reads. The `REPEATABLE READ` isolation level prevents statements from reading data that has been modified by other transactions but not yet committed, and it also prevents other transactions from modifying data that has been read by the current transaction until the current transaction completes.

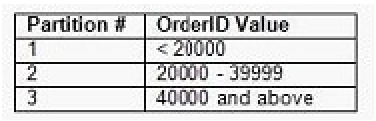
QUESTION 207
You are a database developer. You plan to create a database by using SQL Server 2008. A database contains a table named Sales. The Sales table contains customer order summary information.
You create a stored procedure that uses a SELECT statement. At the moment of execution, the procedure must return a precise summation of the total sales for the current day. You need to use a query hint to prevent any data modification in the Sales table when the stored procedure is being executed.
Which query hint should you recommend?

A. READPAST
B. HOLDLOCK
C. TABLOCKX
D. READCOMMITTED

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 208
You are a database developer on an instance of SQL Server 2008. You have a **SalesHistory** table partitioned as follows:



Partition #	OrderID Value
1	< 20000
2	20000 - 39999
3	40000 and above

You currently have an index on the **SalesHistory** table that implements row-level compression for partition #1 of the index. You want to optimize the index further by implementing page-level compression for the other partitions of the index. Which action should you take?

A. Rebuild the partition function and the **SalesHistory** table.
B. Alter the index to use different compression settings for individual partitions.
C. Implement a partitioned view.
D. Create a filtered index for each partition.

Answer: B
Section: (none)

Explanation/Reference:
In this scenario, you have a partitioned index on the **SalesHistory** table that implements row compression for partition #1 of the index. Partitioned tables and indexes can use different types of compression for individual partitions. You can modify these compression settings using the `ALTER TABLE` and `ALTER INDEX` statements. The `ON PARTITIONS` clause specifies the partition(s) to which settings apply. For example, in this scenario, suppose you used the following statement to create the existing partitioned index:

```
CREATE CLUSTERED INDEX IX_SalesHistoryOrderIDON SalesHistory (OrderID)WITH
(DATA_COMPRESSION = ROW ON PARTITIONS (1),DATA_COMPRESSION = NONE ON PARTITIONS
(2 TO 3));
```

The index would use row compression for partition #1 of the index, and no compression for the other partitions. You could use the following statement to modify the partitioned index to use page compression for partitions #2 and #3:

```
ALTER INDEX IX_SalesHistoryOrderID ON SalesHistoryREBUILD PARTITION = ALL WITH
(DATA_COMPRESSION = PAGE ON PARTITIONS(2 TO 3));
```

Row compression is used to minimize storage at the row level by compressing fixed-length data types. Page compression is used to minimize storage for redundant data stored in pages.
You should not rebuild the partition function and the **SalesHistory** table. There is no need to rebuild the partition function. The partition function defines the boundary values for a partitioned table. In this scenario, the partition boundaries were not changed.
You should not implement a partitioned view. Partitioned views are used when you have similar data stored in multiple tables and want to create a view to allow access to all of the data as if it were stored in a single table. Partitioned views, especially distributed partitioned views that access data across multiple servers, can significantly improve scalability. Partitioned views are implemented by creating a view that queries several tables and combines the results using the `UNION ALL` operator. Although in some situations a partitioned view is helpful, using one would not be applicable in this scenario.
You should not create a filtered index for each partition. A filtered index is a nonclustered index that is defined including a specific `WHERE` clause to optimize the index for specific queries. The index uses the `WHERE` clause condition to index only specific rows in the table. Using a filtered index can improve performance in situations in which specific subsets of the data are frequently accessed, and can reduce the space required for the index. However, in this scenario, a filtered index would not be applicable.

QUESTION 209
You have a table with a string column. A lot of data is duplicated. Need to optimize a stored procedure for performance with the following criteria

- stored procedure takes a string parameter as input
- stored procedure must be optimized for all input params
- only one SELECT statement uses the string input parameter

A. RECOMPILE query hint on the SELECT statement
B. sp_reconfigure
C. sp_recompile
D. alter stored procedure WITH RECOMPILE option

Answer: A
Section: (none)

Explanation/Reference:

QUESTION 210
You are a database administrator on an instance of SQL Server 2008. You are working closely with a group of developers who are developing in a .NET Framework language. The developers are creating user-defined functions that will be used from managed code.
You have enabled your database for CLR integration. You want to use one of the functions directly from your SQL Server instance. The function accepts an input parameter and returns a result set to the caller, and has been marked with the appropriate attributes to access it from SQL Server.
Which action(s) must you take to be able to reference the function from within the `FROM` clause of a `SELECT` statement?

A. Register the assembly with a permission set of `SAFE`.

B. Register the assembly using the default permission set and create the function with the `WITH SCHEMABINDING` clause.
C. Register the assembly with the required permission set and create the function with an `EXTERNAL NAME` clause.
D. Use **sp_configure** to set the **clr enabled** database option to 0.

Answer: C
Section: (none)

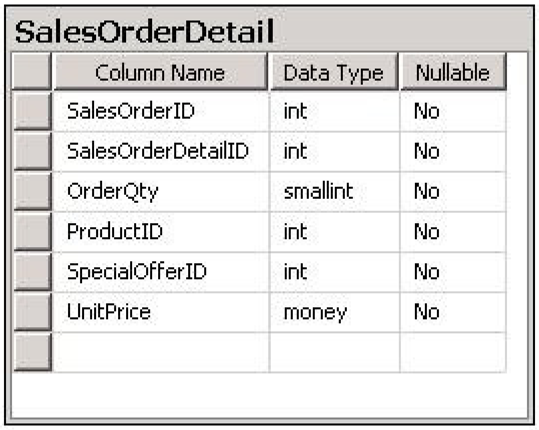
Explanation/Reference:
To reference the function from within the `FROM` clause of a `SELECT` statement, you must register the assembly with the required permission set and create the function with an `EXTERNAL NAME` clause. CLR integration allows you to register and use a function previously created in a .NET Framework language directly from SQL Server. After the function is created in a .NET Framework language and marked with the appropriate attributes to access it from SQL Server, you must take the following actions:
Register the assembly using the `CREATE ASSEMBLY` statement.
Create a function using the `CREATE FUNCTION` statement that can be used to reference the appropriate . NET UDF.

When you register the assembly, you specify a permission set that identifies the permissions that the function should have when it is executed from SQL Server. You may specify one of the following permission sets, depending on the actions the function should be allowed to perform:
`SAFE`: Provides access to the current database, but does not allow the function to access external files, the registry, or environment variables, or to call unmanaged code.
`EXTERNAL_ACCESS`: Provides access to the current database and external resources, including the file system, registry, environment variables, and unmanaged code.
`UNSAFE`: Provides full trust access, and the CLR does not perform any permission checks.

For example, in this scenario, you might use the following statements to register an assembly and create a function that returns a **table** data type:
```
CREATE ASSEMBLY MyAssemblyFROM 'C:\Test.dll' WITH PERMISSION_SET =
EXTERNAL_ACCESS; GO CREATE FUNCTION MyFunction (@parm int)RETURNS TABLE (column1
nvarchar(max), column2 int, column3 datetime)AS EXTERNAL NAME AssemblyName.
NamespaceName.ClassName.FunctionName;
```
In this scenario, you had already enabled CLR integration for the database. Therefore, authorized users would be able to access the CLR function. Because in this scenario the function returns a result set, users could access it in the `FROM` clause of a `SELECT` statement.
The option that states you must register the assembly with a permission set of `SAFE` is incorrect. Although `SAFE` is the default and recommended value, you could register the assembly with the permission set required for the function to perform its defined tasks. However, in this scenario, no specific permission requirements were specified, and performing this action would not allow you to reference the CLR UDF. In addition, you must also create a function to be able to reference the CLR UDF.

The option that states you must register the assembly using the default permission set and create the function with the `WITH SCHEMABINDING` clause is incorrect. The `WITH SCHEMABINDING` clause ensures that no changes can be made to dependent objects that might make a function unusable. However, in this scenario, you must use the `EXTERNAL NAME` clause to identify and reference the CLR UDF.
The option that states you must use **sp_configure** to set the **clr enabled** database option to 0 is incorrect because this disables CLR integration for the database. To use CLR procedures and functions, this database option must be set to 1. In this scenario, you have already enabled the database for CLR integration. Therefore, you do not have to modify this database option.

QUESTION 211
You are a database developer on an instance of SQL Server 2008. You have a **SalesOrderDetail** table defined as follows:



SalesOrderDetail

Column Name	Data Type	Nullable
SalesOrderID	int	No
SalesOrderDetailID	int	No
OrderQty	smallint	No
ProductID	int	No
SpecialOfferID	int	No
UnitPrice	money	No

You want to display product information for products ordered through special promotional offers. You execute the following statement:

```
SELECT SpecialOfferID, ProductID, SUM(OrderQty) AS Total
FROM SalesOrderDetail
GROUP BY GROUPING SETS((ProductID), (ProductID, SpecialOfferID), ())
HAVING GROUPING_ID(ProductID) = 0;
```

When reviewing the result set returned, you notice there are no grand totals displayed.
Which action should you take to display the same result including a grand total of products ordered?

A. Remove the empty grouping set from the `GROUPING SETS` clause.
B. Remove the `HAVING` clause from the query.
C. Modify the `HAVING` clause to check for a value of 1 returned by the `GROUPING_ID` function.
D. Remove the `GROUP BY` and `HAVING` clauses and use `GROUP BY SpecialOfferID WITH ROLLUP` in the `GROUP BY` clause.

Answer: B
Section: (none)

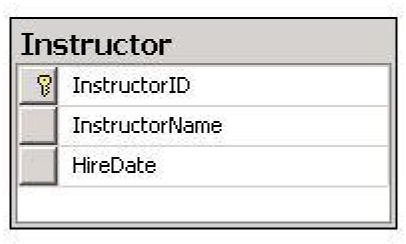
Explanation/Reference:
In this scenario, you included a `GROUPING SETS` clause with the `GROUP BY` clause. The `GROUPING SETS` clause allows you to explicitly specify the groups for which aggregate information should be displayed. This allows you to use
more than one grouping within a single query. The syntax of the `GROUP BY` clause with a `GROUPING SETS` clause is as follows:

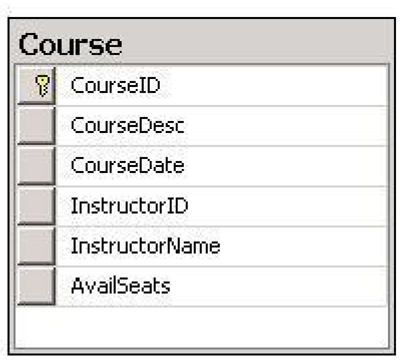
```
GROUP BY GROUPING SETS (groupingset1 [,...groupingsetn])
```

Each grouping set can contain one or more columns or an empty set. Aggregate rows are returned in the result set for only the specified groups.
Specifying an empty set, with `()`, indicates that a grand total row should also be returned in the result set. In this statement, you specified an empty
set in the `GROUPING SETS` clause. However, you used the `GROUPING_ID` function in the `HAVING` clause. The `GROUPING_ID` function returns
either 0 or a 1 to identify the level of grouping. This `HAVING` clause suppresses the grand total rows.
You can also use the `CUBE` and `ROLLUP` operators to aggregate data. The `ROLLUP` operator groups the selected rows in the result set based on the
values in the `GROUP BY` clause and returns one row as a summary row for each group. The `ROLLUP` operator can be used to generate totals and
subtotals. Using the `ROLLUP` operator, a row containing the subtotal and the total is also returned in the result set. When you specify `WITH CUBE`,
a summary row is included in the result set for each possible combination of the columns specified in the `GROUP BY` clause.
When using `ROLLUP`, `CUBE`, or `GROUPING SETS`, aggregate rows can be identified by the `NULL` values.

Summary rows for grand totals will contain
`NULL` values for all grouping columns. If you grouped using two columns, such as **ProductID** and **SpecialOfferID** in this scenario, you could identify
the aggregate rows as follows:
 Summary rows that represent totals for each **ProductID** would have a value for **ProductID** and a `NULL` value for **SpecialOfferID**.
 Summary rows that represent totals for each **SpecialOfferID** would have a value for **SpecialOfferID** and a `NULL` value for **ProductID**.
You should not remove the empty grouping set from the `GROUPING SETS` clause. To include grand totals in a query that uses the `GROUPING SETS`
clause, you must include an empty set.
You should not modify the `HAVING` clause to check for a value of 1 returned by the `GROUPING_ID` function. This would return only the grand total
row, and filter out the aggregate rows for products and the aggregate rows for unique **ProductID** and **SpecialOfferID** combinations.
You should not remove the `GROUP BY` and `HAVING` clauses and use `GROUP BY SpecialOfferID WITH ROLLUP` in the `GROUP BY` clause. All
columns in the `SELECT` list must either use an aggregate function or be included in the query's `GROUP BY` clause.

QUESTION 212
You are a database developer on an instance of SQL Server 2008. Your database is used to store information about training courses and
instructors. Each training course has a single instructor. You have defined the **Course** and **Instructor** entities as follows:





You want to normalize the data model.
What should you do?

A. Remove the **InstructorName** attribute from the **Course** entity.
B. Remove the **InstructorID** attribute from the **Course** entity.
C. Add the **CourseID** attribute to the **Instructor** entity.
D. Remove the **Instructor** entity.

Answer: A
Section: (none)

Explanation/Reference:
A table should only contain attributes, or columns, related to a single
entity. Attempting to store too much data in a table can cause data management problems. With the given entities, it would be possible to enter a
different **InstructorName** in the two tables. In this scenario, the **Course** entity should only contain information

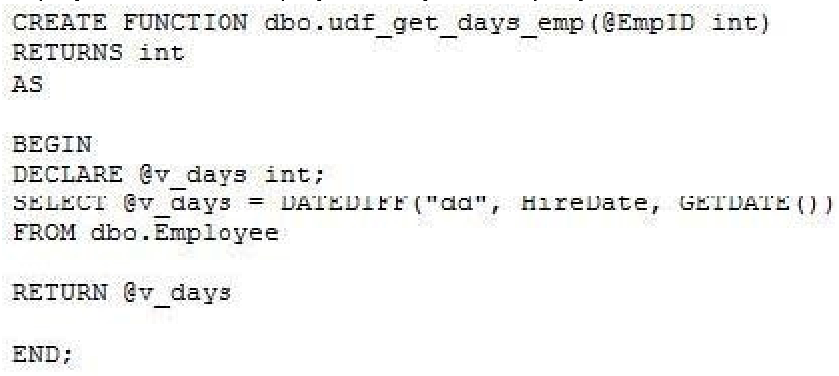
for each course offered, and the
Instructor entity should only contain information for each instructor. The **InstructorName** attribute should be included only in the **Instructor** table,
not in the **Course** table. To remove the **InstructorName** column from the **Course** entity, you could use the following statement:

```
ALTER TABLE Course DROP COLUMN InstructorName;
```

You should not remove the **Instructor** entity. The **Instructor** entity should remain because it contains all columns pertaining to each instructor.
You should not add the **CourseID** column to the **Instructor** entity. Because you have already associated the **InstructorID** column with the **Course**
entity, it is not necessary to associate the **CourseID** column with the **Instructor** entity. This can be accomplished by creating a `FOREIGN KEY`
constraint in the **Course** table that references the **InstructorID** in the **Instructor** table, or by using joins.
You should not remove the **InstructorID** column from the **Course** entity. The **InstructorID** column must exist in the **Course** table to associate each
course with a specific instructor. To do so, you can use a foreign key or a join to associate the two tables. For example, in this scenario, you could
use the following Transact-SQL to create a `FOREIGN KEY` constraint to associate the **Instructor** and **Course** tables:

```
ALTER TABLE dbo.Course
ADD FOREIGN KEY (InstructorID) REFERENCES dbo.Instructor(InstructorID);
```

QUESTION 213
You are a database developer on an instance of SQL Server 2008. Your **Employee** table contains a **HireDate** column that contains the date each employee was employed with the company.
You have created the user-defined function shown in the exhibit to calculate the number days that an employee has been employed with your company:



```
CREATE FUNCTION dbo.udf_get_days_emp(@EmpID int)
RETURNS int
AS

BEGIN
DECLARE @v_days int;
SELECT @v_days = DATEDIFF("dd", HireDate, GETDATE())
FROM dbo.Employee

RETURN @v_days

END;
```

You have some complex queries that use the **dbo.udf_get_days_emp** function to return a value. You want to optimize performance for these queries with the least amount of effort.
Which action should you take?

A. Remove the reference to the `GETDATE()` built-in function in the function's definition.
B. Replace the function with a stored procedure that accepts an input variable and returns an output variable.

C. Create a view that includes only the function's result and create an index on the view.
D. Create a computed column in the **Employee** table that uses the function in its expression definition, and create an index on the computed column.

Answer: B
Section: (none)

Explanation/Reference:
In this scenario, the **dbo.udf_get_days_emp** function is a nondeterministic function. A function is deterministic if it returns the same value each time it is passed the same values. In this scenario, the function could return different values even if it was passed the same input, such as when `GETDATE()` returns a different value. Nondeterministic user-defined functions have certain restrictions on how they can be used. In this scenario, the best approach would be to rewrite the function as a stored procedure because stored procedures do not have this restriction. If you are not sure whether a function is deterministic, you can return the `IsDeterministic` property using the `OBJECTPROPERTY` function. The `IsDeterministic` property is 1 if the function is deterministic, and 0 if it is not. For example, you could use the following query to determine whether the `dbo.udf_get_days_emp`function is deterministic or nondeterministic:

```
SELECT OBJECTPROPERTY(OBJECT_ID('dbo.udf_get_days_emp'), 'IsDeterministic');
```

You should not remove the reference to the `GETDATE()` built-in function in the function's definition. Although this would make the function deterministic, you would not be able to calculate the number of days the employee has been employed.
You should not create a view that includes only the function's result and create an index on the view because you cannot create a view that references a nondeterministic function. In this scenario, you would have to create a view based on each value returned from the function. Unlike a UDF or stored procedure, you cannot pass a parameter to a view.
You should not create a computed column in the **Employee** table that uses the function in its expression definition and create an index on the computed column. Computed columns are virtual columns that are not physically stored in the table by default. Each computed column uses the `AS` keyword followed by an expression that evaluates to a value. The expression can contain constants, functions, operators, and references to other columns within the table. The value of the computed column is calculated each time a query that references it executes. You can also include the optional `PERSISTED` keyword when creating a computed column. When a persisted computed column is created, it is physically stored in the table and is recalculated each time a column value referenced in the calculation expression is changed. To be able to create an index on a computed column, the column must be deterministic and precise. A computed column is considered deterministic if it produces the same value each time it is passed the same values. A computed column is considered precise if it does not perform floating-point calculations using either a `float` or `real` data type. In this scenario, the function is nondeterministic, so using it in the computed column's expression would prevent you from creating an index on the computed column. You can query the `IsDeterministic` and `IsPrecise` properties using the `COLUMNPROPERTY` function to determine if a computed column is deterministic and precise, respectively.

QUESTION 214
You have a stored procedure that uses a SELECT statement to retrieve a sample data set from a table. It has been determined that the query is taking too long to return data.

Given that some inconsistent data can be allowed to appear. What type of isolation should be used for the query to return results fast?

A. Repeatable Read
B. Read Uncommited
C. ReadCommitted
D. ReadCommitted Snapshot

Answer: B
Section: (none)

Explanation/Reference:
READ UNCOMMITTED

Specifies that statements can read rows that have been modified by other transactions but not yet committed.

Transactions running at the READ UNCOMMITTED level do not issue shared locks to prevent other transactions from modifying data read by the current transaction. READ UNCOMMITTED transactions are also not blocked by exclusive locks that would prevent the current transaction from reading rows that have been modified but not committed by other transactions. When this option is set, it is possible to read uncommitted modifications, which are called dirty reads. Values in the data can be changed and rows can appear or disappear in the data set before the end of the transaction. This option has the same effect as setting NOLOCK on all tables in all SELECT statements in a transaction. This is the least restrictive of the isolation levels.

QUESTION 215
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 instance to create a database solution which has some requirements: mport data from various data sources such as Microsoft Office Excel, Microsoft SQL Server 2000, Microsoft SQL Server 2005, and CSV files; Profile the source data before it is imported; Allow mobile users to use heterogeneous data stores; Provide mobile users with collaboration and offline capabilities. In order to achieve these requirements, you have to configure the suitable SQL Server components. When performing this, you have to use as little administrative effort as possible. So which SQL Server components should you use? (choose more than one)

A. Reporting Services
B. Integration Services
C. Analysis Services
D. Notification Services
E. Microsoft Sync Framework

Answer: BE
Section: (none)

Explanation/Reference:

QUESTION 216
You work for a company that builds parts for an aerospace contractor. You use the following Transact-SQL to update the **Widget** table in the **Research** database:

```
USE Research GO
BEGIN TRANSACTION ToleranceUpdate WITH MARK 'UPDATE Tolerance level';
GO
 UPDATE Engineering.WidgetSET Tolerance = Tolerance * 1.03
  WHERE WidgetID LIKE 'AA-%';GO
COMMIT TRANSACTION ToleranceUpdate;
GO
```

After a few hours, you notice that an error occurred in the database. A regular backup and log backup of the **Research** database have been made. You restore the full backup of the **Research** database.
What should you do restore the **Widget** table to the `ToleranceUpdate` transaction?

A. Restore the log backup using `WITH STOPBEFOREMARK='ToleranceUpdate'`
B. Restore the log backup using `WITH STOPATMARK='ToleranceUpdate'`
C. Use the `ROLLBACK` statement to roll back to the `ToleranceUpdate` transaction.
D. Restore the transaction log. Use a `ROLLBACK` statement to roll back to the `ToleranceUpdate` transaction.

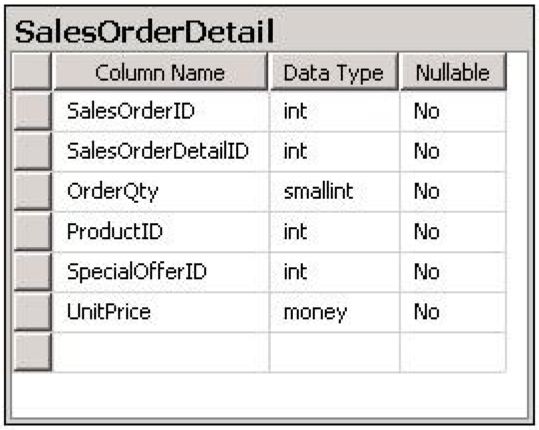
Answer: B
Section: (none)

Explanation/Reference:
In this scenario, you marked the transaction with the name **ToleranceUpdate**. This will allow you to perform point-in-time recovery at the transaction level. The `STOPATMARK` and `STOPATBEFOREMARK` clauses of the `RESTORE` statement are used to restore the database to the saved point in the transaction or to a point before the transaction began, respectively. You could use the following statement to restore the log:

```
RESTORE LOG Research FROM ResearchBackUpWITH RECOVERY,
STOPATMARK='ToleranceUpdate';
```

When restoring a database, you must restore the latest full backup. If applicable, you must restore the latest differential backup since the last full backup. You should then restore each transaction log backup that was made since the last full backup or, if you made differential backups, the transaction log backup that was made since the last differential backup. You should also make a copy of the current transaction log. This backup will be called the tail-log backup because it contains the transactions that have not been recorded.
When you restore the latest full database backup and the latest differential backup, specify `WITH NORECOVERY` in the `RESTORE` statement. The `NORECOVERY` option is required in this scenario because you will be applying transaction log backups after the database is restored. Then, you consecutively restore each of the transaction log backups that were made after the last full database backup or differential backup. Each transaction log backup must be restored by specifying the `NORECOVERY` option so that all subsequent transaction log backups can be restored. You should restore the transaction log backup that was made on the active transaction log. This is the latest log backup.
When restoring the transaction log, you should use the `STOPATMARK` option along with the `RECOVERY` option in the `RESTORE LOG` statement. The `STOPAT` option specifies the desired point in time to which you want to restore the database, and the `WITH RECOVERY` option brings the database to a consistent state. You can use marked transactions across databases to recover multiple databases to the same specific point, but doing so causes any transactions committed after the mark used as the recovery point to be lost.
You should not restore the log backup using `WITH STOPBEFOREMARK='ToleranceUpdate'`. The `WITH STOPBEFOREMARK` clause in the `RESTORE LOG` statement uses the log record before the saved point as the mark for recovery. In this scenario, you wanted to restore the **Widget** table to the `ToleranceUpdate` transaction, not to a point before it started.
You should not use a `ROLLBACK` statement to roll back to the `ToleranceUpdate` transaction. You can use a `ROLLBACK` statement to roll back to a specific point in a transaction, but to do so, you must first use the `SAVE TRANSACTION` statement to create a named savepoint. In this scenario, you did not create a savepoint. You created a marked transaction.
You should not restore the transaction log and use a `ROLLBACK` statement to roll back to the `ToleranceUpdate` transaction. You must create a savepoint to roll back to a specific point in a transaction.

QUESTION 217
You are a database developer on an instance of SQL Server 2008. You have a **SalesOrderDetail** table defined as follows:



SalesOrderDetail

Column Name	Data Type	Nullable
SalesOrderID	int	No
SalesOrderDetailID	int	No
OrderQty	smallint	No
ProductID	int	No
SpecialOfferID	int	No
UnitPrice	money	No

You want to display product information for products ordered through special promotional offers. You execute the following statement:

```
SELECT SpecialOfferID, ProductID, SUM(OrderQty) AS Total
FROM SalesOrderDetail
GROUP BY GROUPING SETS((ProductID), (ProductID, SpecialOfferID), ())
HAVING GROUPING_ID(ProductID) = 0;
```

When reviewing the result set returned, you notice there are no grand totals displayed.
Which action should you take to display the same result including a grand total of products ordered?

A. Remove the empty grouping set from the `GROUPING SETS` clause.

B. Remove the `HAVING` clause from the query.

C. Modify the `HAVING` clause to check for a value of 1 returned by the `GROUPING_ID` function.

D. Remove the `GROUP BY` and `HAVING` clauses and use `GROUP BY SpecialOfferID WITH ROLLUP` in the `GROUP BY` clause.

Answer: B
Section: (none)

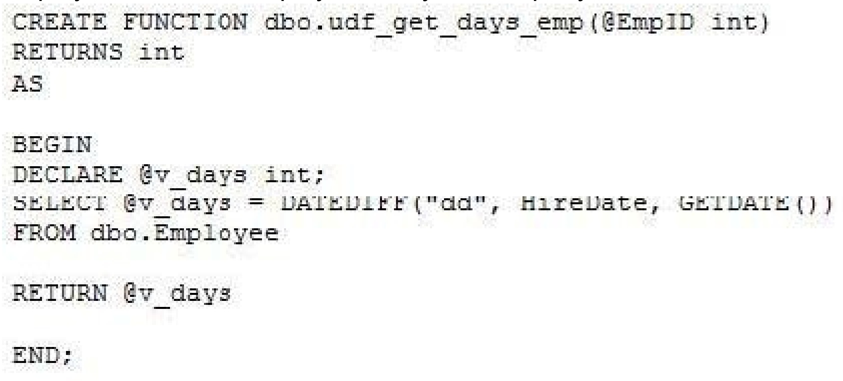
Explanation/Reference:
In this scenario, you included a `GROUPING SETS` clause with the `GROUP BY` clause. The `GROUPING SETS` clause allows you to explicitly specify the groups for which aggregate information should be displayed. This allows you to use
more than one grouping within a single query. The syntax of the `GROUP BY` clause with a `GROUPING SETS` clause is as follows:

```
GROUP BY GROUPING SETS (groupingset1 [,...groupingsetn])
```

Each grouping set can contain one or more columns or an empty set. Aggregate rows are returned in the result set for only the specified groups.
Specifying an empty set, with `()`, indicates that a grand total row should also be returned in the result set. In this statement, you specified an empty
set in the `GROUPING SETS` clause. However, you used the `GROUPING_ID` function in the `HAVING` clause. The `GROUPING_ID` function returns
either 0 or a 1 to identify the level of grouping. This `HAVING` clause suppresses the grand total rows.
You can also use the `CUBE` and `ROLLUP` operators to aggregate data. The `ROLLUP` operator groups the selected rows in the result set based on the
values in the `GROUP BY` clause and returns one row as a summary row for each group. The `ROLLUP` operator can be used to generate totals and
subtotals. Using the `ROLLUP` operator, a row containing the subtotal and the total is also returned in the result set. When you specify `WITH CUBE`,
a summary row is included in the result set for each possible combination of the columns specified in the `GROUP BY` clause.
When using `ROLLUP`, `CUBE`, or `GROUPING SETS`, aggregate rows can be identified by the `NULL` values. Summary rows for grand totals will contain
`NULL` values for all grouping columns. If you grouped using two columns, such as **ProductID** and **SpecialOfferID** in this scenario, you could identify
the aggregate rows as follows:
 Summary rows that represent totals for each **ProductID** would have a value for **ProductID** and a `NULL` value for **SpecialOfferID**.
 Summary rows that represent totals for each **SpecialOfferID** would have a value for **SpecialOfferID** and a `NULL` value for **ProductID**.
You should not remove the empty grouping set from the `GROUPING SETS` clause. To include grand totals in a query that uses the `GROUPING SETS`
clause, you must include an empty set.
You should not modify the `HAVING` clause to check for a value of 1 returned by the `GROUPING_ID` function. This would return only the grand total
row, and filter out the aggregate rows for products and the aggregate rows for unique **ProductID** and

SpecialOfferID combinations.
You should not remove the `GROUP BY` and `HAVING` clauses and use `GROUP BY SpecialOfferID WITH ROLLUP` in the `GROUP BY` clause. All
columns in the `SELECT` list must either use an aggregate function or be included in the query's `GROUP BY` clause.

QUESTION 218
You are a database developer on an instance of SQL Server 2008. Your **Employee** table contains a **HireDate** column that contains the date each employee was employed with the company.
You have created the user-defined function shown in the exhibit to calculate the number days that an employee has been employed with your company:



```
CREATE FUNCTION dbo.udf_get_days_emp(@EmpID int)
RETURNS int
AS

BEGIN
DECLARE @v_days int;
SELECT @v_days = DATEDIFF("dd", HireDate, GETDATE())
FROM dbo.Employee

RETURN @v_days

END;
```

You have some complex queries that use the **dbo.udf_get_days_emp** function to return a value. You want to optimize performance for these queries with the least amount of effort.
Which action should you take?

A. Remove the reference to the `GETDATE()` built-in function in the function's definition.
B. Replace the function with a stored procedure that accepts an input variable and returns an output variable.

C. Create a view that includes only the function's result and create an index on the view.
D. Create a computed column in the **Employee** table that uses the function in its expression definition, and create an index on the computed column.

Answer: B
Section: (none)

Explanation/Reference:
In this scenario, the **dbo.udf_get_days_emp** function is a nondeterministic function. A function is deterministic if it returns the same value each time it is passed the same values. In this scenario, the function could return different values even if it was passed the same input, such as when `GETDATE()` returns a different value. Nondeterministic user-defined functions have certain restrictions on how they can be used. In this scenario, the best approach would be to rewrite the function as a stored procedure because stored procedures do not have this restriction. If you are not sure whether a function is deterministic, you can return the `IsDeterministic` property using the `OBJECTPROPERTY` function. The `IsDeterministic` property is 1 if the function is deterministic, and 0 if it is not. For example, you could use the following query to determine whether the `dbo.udf_get_days_emp` function is deterministic or nondeterministic:
```
SELECT OBJECTPROPERTY(OBJECT_ID('dbo.udf_get_days_emp'), 'IsDeterministic');
```

You should not remove the reference to the `GETDATE()` built-in function in the function's definition. Although this would make the function deterministic, you would not be able to calculate the number of days the employee has been employed.
You should not create a view that includes only the function's result and create an index on the view because you cannot create a view that references a nondeterministic function. In this scenario, you would have to

create a view based on each value returned from the function. Unlike a UDF or stored procedure, you cannot pass a parameter to a view.
You should not create a computed column in the **Employee** table that uses the function in its expression definition and create an index on the computed column. Computed columns are virtual columns that are not physically stored in the table by default. Each computed column uses the `AS` keyword followed by an expression that evaluates to a value. The expression can contain constants, functions, operators, and references to other columns within the table. The value of the computed column is calculated each time a query that references it executes. You can also include the optional `PERSISTED` keyword when creating a computed column. When a persisted computed column is created, it is physically stored in the table and is recalculated each time a column value referenced in the calculation expression is changed. To be able to create an index on a computed column, the column must be deterministic and precise. A computed column is considered deterministic if it produces the same value each time it is passed the same values. A computed column is considered precise if it does not perform floating-point calculations using either a `float` or `real` data type. In this scenario, the function is nondeterministic, so using it in the computed column's expression would prevent you from creating an index on the computed column. You can query the `IsDeterministic` and `IsPrecise` properties using the `COLUMNPROPERTY` function to determine if a computed column is deterministic and precise, respectively.

QUESTION 219
You work for a company that builds parts for an aerospace contractor. You use the following Transact-SQL to update the **Widget** table in the **Research** database:
```
USE Research GO BEGIN TRANSACTION ToleranceUpdateWITH MARK 'UPDATE Tolerance
level';GO
UPDATE Engineering.WidgetSET Tolerance = Tolerance * 1.03 WHERE WidgetID LIKE
'AA-%';GO
COMMIT TRANSACTION ToleranceUpdate;GO
```
After a few hours, you notice that an error occurred in the database. A regular backup and log backup of the **Research** database have been made. You restore the full backup of the **Research** database.
What should you do restore the **Widget** table to the `ToleranceUpdate` transaction?

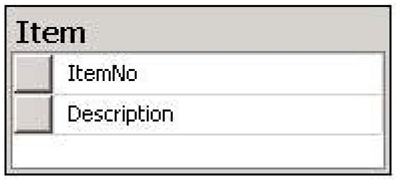
A. Restore the log backup using `WITH STOPBEFOREMARK='ToleranceUpdate'`
B. Restore the log backup using `WITH STOPATMARK='ToleranceUpdate'`
C. Use the `ROLLBACK` statement to roll back to the `ToleranceUpdate` transaction.
D. Restore the transaction log. Use a `ROLLBACK` statement to roll back to the `ToleranceUpdate` transaction.

Answer: B
Section: (none)

Explanation/Reference:
In this scenario, you marked the transaction with the name **ToleranceUpdate**. This will allow you to perform point-in-time recovery at the transaction level. The `STOPATMARK` and `STOPATBEFOREMARK` clauses of the `RESTORE` statement are used to restore the database to the saved point in the transaction or to a point before the transaction began, respectively. You could use the following statement to restore the log:
```
RESTORE LOG Research FROM ResearchBackUpWITH RECOVERY,
STOPATMARK='ToleranceUpdate';
```
When restoring a database, you must restore the latest full backup. If applicable, you must restore the latest differential backup since the last full backup. You should then restore each transaction log backup that was made since the last full backup or, if you made differential backups, the transaction log backup that was made since the last differential backup. You should also make a copy of the current transaction log. This backup will be called the tail-log backup because it contains the transactions that have not been recorded.
When you restore the latest full database backup and the latest differential backup, specify `WITH NORECOVERY` in the `RESTORE` statement. The `NORECOVERY` option is required in this scenario because you will be applying transaction log backups after the database is restored. Then, you consecutively restore each of the transaction log backups that were made after the last full database backup or differential backup. Each

transaction log backup must be restored by specifying the `NORECOVERY` option so that all subsequent transaction log backups can be restored. You should restore the transaction log backup that was made on the active transaction log. This is the latest log backup.
When restoring the transaction log, you should use the `STOPATMARK` option along with the `RECOVERY` option in the `RESTORE LOG` statement. The `STOPAT` option specifies the desired point in time to which you want to restore the database, and the `WITH RECOVERY` option brings the database to a consistent state. You can use marked transactions across databases to recover multiple databases to the same specific point, but doing so causes any transactions committed after the mark used as the recovery point to be lost.
You should not restore the log backup using `WITH STOPBEFOREMARK='ToleranceUpdate'`. The `WITH STOPBEFOREMARK` clause in the `RESTORE LOG` statement uses the log record before the saved point as the mark for recovery. In this scenario, you wanted to restore the **Widget** table to the `ToleranceUpdate` transaction, not to a point before it started.
You should not use a `ROLLBACK` statement to roll back to the `ToleranceUpdate` transaction. You can use a `ROLLBACK` statement to roll back to a specific point in a transaction, but to do so, you must first use the `SAVE TRANSACTION` statement to create a named savepoint. In this scenario, you did not create a savepoint. You created a marked transaction.
You should not restore the transaction log and use a `ROLLBACK` statement to roll back to the `ToleranceUpdate` transaction. You must create a savepoint to roll back to a specific point in a transaction.

QUESTION 220
You have a **Test** database that resides on an instance of SQL Server. The **Test** database contains the **Item** table, which includes the **ItemNo** and **Description** columns as follows:



The table is currently empty, and you execute the following Transact-SQL code:
```
BEGIN TRANSACTION
INSERT INTO Item VALUES (1, 'Computer')
INSERT INTO Item VALUES (2, 'Printer')
DELETE FROM Item WHERE ItemNo = 1
SAVE TRANSACTION SaveA
INSERT INTO Item VALUES (3, 'Router')
INSERT INTO Item VALUES (4, 'Monitor')
SAVE TRANSACTION SaveB
INSERT INTO Item VALUES (5, 'Keyboard')
UPDATE Item SET Description = 'Wireless Keyboard' WHERE ItemNo = 5
ROLLBACK TRANSACTION SaveA
UPDATE Item SET Description = 'Mouse' WHERE ItemNo = 4
COMMIT TRANSACTION
```

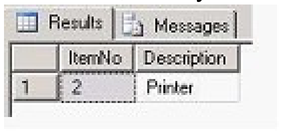
Which item description will be in the **Item** table?

A. Router
B. Wireless Keyboard
C. Printer
D. Mouse

Answer: C
Section: (none)

Explanation/Reference:

. In this scenario, you executed Transact-SQL code that uses transactional savepoints. Savepoints allow you to roll back portions of transactions as needed. To create a savepoint to which you can roll back, you use the `SAVE TRANSACTION` statement and specify a savepoint name. Then, when you issue a `ROLLBACK` statement, you can specify the savepoint to which you want to roll back. In this scenario, you issued two `INSERT` statements to insert the first two rows and a `DELETE` statement that deletes the first inserted row. Then, you issued a `SAVE TRANSACTION` statement to create a savepoint named **SaveA**. Next, you inserted rows 3 and 4, and created another savepoint named **SaveB**. Having multiple savepoints allows you to roll back to different places within the same transaction. Next, you inserted another row in the table, and updated the row with **ItemNo** 5. Then, you issued a `ROLLBACK TRANSACTION` statement specifying `SaveA`. This statement rolled back all the changes that had occurred since creating the **SaveA** savepoint.
After the rollback, you issued an update to **ItemNo** 4. However, the `INSERT` for this row was previously rolled back. Therefore, no update occurred.
Therefore, after you commit the transaction, only the second row inserted into the table will remain as shown:



	ItemNo	Description
1	2	Printer

You should note that when rolling back to a savepoint, SQL Server holds resources until the entire transaction is either committed or rolled back. Therefore, you should always issue a `COMMIT` or a complete `ROLLBACK` even if you have previously rolled back part of the transaction.

QUESTION 221
You are designing a database that will be used for reporting purposes. You need to minimize the data storage requirements and improve the application response time.
What should you recommend?

A. row compression
B. sparse columns
C. table partitioning
D. XML

Answer: A
Section: (none)

Explanation/Reference:

QUESTION 222
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. multilingual data and a table that has 100 million rows will be stored in the database. There're 1,000 columns in the table. The columns are on the basis of the nvarchar(max) data type. Only 3 percent of the rows will be populated for each column. You have to optimize storage space by designing the table. So what should you do?

Certkey.com - Make You Succeed To Pass IT Exams
Certkey 70-451

A. Row compression should be used
B. The columns should be defined as sparse columns
C. The column data type should be changed to varchar(max)
D. Minimize the disk space used by using NTFS file system compression

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 223
You are a database developer. You plan to design a database solution by using SQL Server 2008.
You plan to design a complex multi-statement stored procedure in the following manner.
CREATE PROCEDURE Sales.GetCustomerActivity
@StartDate datetime
AS
SELECT order_id, order_date, customer_id
FROM Sales.Orders
WHERE order_date >= @StartDate
...
On testing, you discover that the stored procedure occasionally takes a longer than expected time to execute.
You discover that this degradation is caused by the first statement in the stored procedure.
You need to ensure that the stored procedure is consistently executed in the minimum possible time.
What should you do?

A. Run the EXEC sp_recompile GetCustomerActivity command.
B. Create a plan guide to apply the OPTION (RECOMPILE) clause to the first statement.
C. Modify the stored procedure by adding the WITH RECOMPILE clause.
D. Replace the first statement in the stored procedure with the following Transact-SQL statement.
UPDATE STATISTICS Sales.GetCustomerActivity WITH RESAMPLE Certkey.com - Make You Succeed To Pass IT Exams
Certkey 70-451

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 224
You are a database developer on an instance of SQL Server. You have defined an inline table-valued function using the following statement:
```
CREATE FUNCTION udf_GetOrderDetails (@ordid int)RETURNS TABLE AS RETURN (SELECT
sh.OrderDate, sd.*FROM SalesOrderHeader sh INNER JOIN SalesOrderDetail sd ON sh.
SalesOrderID = sd.SalesOrderID
WHERE sd.SalesOrderID=@ordid);
```
You have table named **SalesOrderHistory** that contains a **SalesOrderID** column. You want to query the **SalesOrderHistory** table and use the **udf_GetOrderDetails** function to return order details. All sales orders in the **SalesOrderHistory** table and the associated order details should be displayed, even if no details exist for a given order.
Which query should you use?

A.
```
SELECT * FROM
FROM SalesOrderHistory h
CROSS APPLY udf_GetOrderDetails (h.SalesOrderID) AS d
```
B.
```
SELECT h.*, d.*FROM SalesOrderHistory h
OUTER APPLY udf_GetOrderDetails (h.SalesOrderID) AS d ORDER BY h.SalesOrderID;
```
C.
```
SELECT * FROM udf_GetOrderDetails(t.SalesOrderID)) as t;
```

D.
```
SELECT h.*, d.*
INNER JOIN udf_GetOrderDetails(h.SalesOrderID) AS d
ON h.SalesOrderID = d.SalesOrderID
ORDER BY h.SalesOrderID;
```

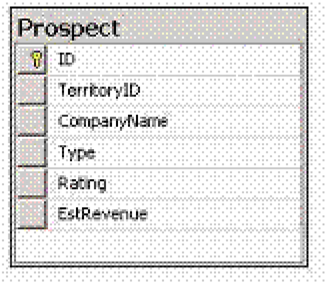
Answer: B
Section: (none)

Explanation/Reference:
You should use the following query:
```
SELECT h.*, d.*FROM SalesOrderHistory hOUTER APPLY udf_GetOrderDetails (h.
SalesOrderID) AS d ORDER BY h.SalesOrderID;
```
In this scenario, you wanted to join the results of a table-valued function to another table. You can accomplish this using the `APPLY` operator in the `FROM` clause of your query. The syntax is as follows:
```
SELECT column_list FROM outer_table {CROSS | OUTER} APPLY inner_table
```
The `inner_table` may specify another table or view, but may also specify a function that returns a **table** type. In this scenario, you specified the **udf_GetOrderDetails** function as the inner table. For each row in the query, the **udf_GetOrderDetails** function is called using the **SalesOrderID** from the outer table. If the function returns rows, these rows are joined with the outer table using`UNION ALL` and are available to the query. You should use `OUTER APPLY` to ensure that all rows in the outer table, **SalesOrderHistory**, are returned even if no order details exist. For rows in the **SalesOrderHistory** table that are not returned by the function, all order detail columns will be assigned a `NULL` value. When using the `APPLY` operator, the `outer_table` may also use a table-valued function, but it cannot pass the function columns to the `inner_table` as arguments.
You should not use the query that includes an inner join because this syntax is invalid. This statement will return the following error:
```
Msg 4104, Level 16, State 1, Line 12The multi-part identifier "h.SalesOrderID"
could not be bound.
```
You should not use the query that uses a subquery because it attempts to reference`t.SalesOrderID` in the inner query. This statement will generate the following error:
```
Msg 4104, Level 16, State 1, Line 25The multi-part identifier "t.SalesOrderID"
could not be bound.
```
You should not use the query that uses`CROSS APPLY`. A `CROSS APPLY` operator only returns rows that had rows returned by the **udf_GetOrderDetails** function. In this scenario, you wanted to return all rows from the **SalesOrderHistory** table, even if they contained no order
details. A `CROSS APPLY` would not return rows in the **SalesOrderHistory** table that did not contain order details.

QUESTION 225
You are a database developer on an instance of SQL Server 2008. Your **Prospect** table is shown in the exhibit. (Click the **Exhibit(s)** button.)
You want to retrieve data from the **Prospect** table in XML format. You want the XML in the result set to meet the following requirements:
A root `<ProspectData>` element should enclose the XML.
Each prospect should be represented with a`<Prospect>` element that includes an `ID` and `Name` attribute.
Each `<Prospect>` element should enclose a `<Detail>` element that includes subelements for the type and rating.

You want to accomplish this with the least amount of effort.
Which query should you execute?

Exhibit:



A.
```
SELECT ID AS "@ID",
       CompanyName AS "@Name",
       Type AS "Detail/Type",
       Rating AS "Detail/Rating"
  FROM Prospect
  FOR XML PATH('Prospect'), Root('ProspectData');
```

B. select
```
    AS Tag,
    NULL AS Parent,
    ID AS [Prospect!1!ID!element],
    CompanyName AS [Prospect!1!Name],
    Type AS [Prospect!1!Type],
    NULL AS [Prospect!1!Rating]
  FROM Prospect
  FOR XML EXPLICIT ROOT('ProspectData');
```

C.
```
SELECT ID,
Type AS Type,
Rating AS Rating
FROM Prospect
FOR XML AUTO, TYPE, ROOT('ProspectData');
```

D.
```
SELECT ID AS "@ID",
Type AS "Type",
Rating AS "Rating"
INTO XMLOutputTbl
FROM Prospect
FOR XML PATH('Prospect'), Root('ProspectData');
```

Answer: A
Section: (none)

Explanation/Reference:
The `FOR XML` clause specifies that the result of the `SELECT` statement should be returned in XML format. You can specify one of the following modes with the `FOR XML` clause:
`RAW`: A single `<row>` element is be generated for each row in the rowset. Each non-null column value generates an attribute with the name identical to the column's name or the column's alias.
`AUTO`: Nested elements are returned using the columns specified in the `SELECT` list. Each non-null column value generates an attribute named according to the column name or column alias. The element nesting is based on the order the columns are specified in the `SELECT` list.
`EXPLICIT`: Each row in the rowset can be defined in detail specifying attributes and elements.
`PATH`: Each row in the rowset can be defined in detail, but column names and column aliases are specified as XPath expressions.

When using `PATH` mode with the `FOR XML` clause, you can define each row in the rowset in detail. In this scenario, you specified the `ROOT` option, which will generate a root element named `<ProspectData>` in the output that will enclose the other elements. You should also specify `PATH ('Prospect')`. This will create a `<Prospect>` element that will enclose the other elements. With `FOR XML PATH`, you can use aliases in the `SELECT` list to further control how the XML is formatted. In this scenario, you specified the **ID** column using the `"@ID"` alias and the **CompanyName** column using the `"@Name"` alias. Because the alias begins with an

ampersand (`@`), this will generate `ID` and `Name` attributes for each `<Prospect>` element. Next, you should specify an alias of `"Detail/Type"` for the **Type** column. Because the alias does not begin with an ampersand (`@`), this generates a `<Detail>` element within the `<Prospect>` element that contains a `<Type>` subelement. Finally, you should specify an alias of `"Detail/Rating"` for the **Rating** column, which generates an additional `<Rating>` subelement within the `<Detail>` element. Using `PATH` mode is less complex than using `EXPLICIT` mode, but provides much more flexibility mapping data to XML than the other modes.
This query would generate XML output similar to the following, with a `<Prospect>` element for each row returned:

```
<ProspectData><Prospect ID="1" Name="VirtuArt"><Detail> <Type>A</Type><Rating>9</
Rating></Detail></Prospect><Prospect ID="2" Name="NuTex Corporation"><Detail>
<Type>C</Type><Rating>3</Rating></Detail></Prospect><Prospect ID="3"
Name="InterConn"><Detail> <Type>A</Type><Rating>7</Rating></Detail>
</Prospect></ProspectData>
```

You should not use the query that specifies the `TYPE` option in the `FOR XML` clause because it does not format the XML as required. When you specify the `TYPE` option, the XML is returned as an **xml** data type instance rather than XML text. This query would output an **xml** data type that represented the data in a format similar to the following:

```
<ProspectData><Prospect ID="1" Name="VirtuArt" Type="A" Rating="9" /><Prospect
ID="2" Name="NuTex Corporation" Type="C" Rating="3" /><Prospect ID="3"
Name="InterConn" Type="A" Rating="7" /></ProspectData>
```

The `TYPE` option can be very useful in some situations, such as if you wanted to nest queries that used `FOR XML`, or if you wanted to assign the result to an **xml** variable to perform some additional processing.
You should not use the query that includes an `INTO` clause in the `SELECT` statement because the `FOR XML` clause cannot be used with query that uses the `INTO` clause. This query would generate the following error message:

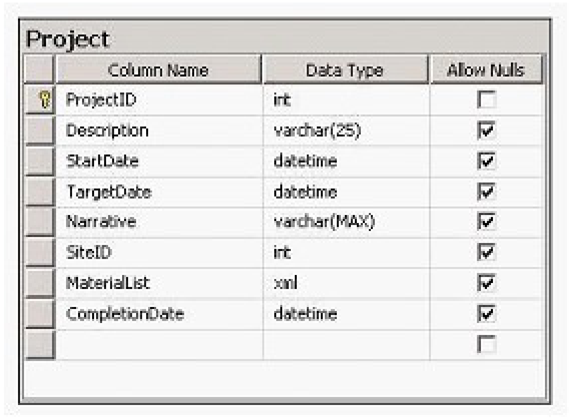
```
Msg 6819, Level 16, State 1, Line 9The FOR XML clause is not allowed in a SELECT
INTO statement.
```

You should not use the query that uses the `FOR XML EXPLICIT` clause. With `FOR XML EXPLICIT`, you can provide details of how the XML result should be formatted, but the syntax is more complex than using `FOR XML PATH`. The first two columns in the `SELECT` statement must specify the **Tag** column and the **Parent** column, which indicate the level of the current tag and the current tag's parent element, respectively. The other columns in the `SELECT`
list are actual columns in the table. The output would be similar to the following:

```
<ProspectData><Prospect Name="VirtuArt" Type="A"><ID>1</ID></Prospect><Prospect
Name="NuTex Corporation" Type="C"><ID>2</ID></Prospect><Prospect Name="InterConn"
Type="A"><ID>3</ID></Prospect></ProspectData>
```

To include nested elements within the output, you would explicitly format each level in a separate `SELECT` statement, specifying a different **Tag** value and the tag's parent, and combine the result sets using the `UNION ALL` operator.

QUESTION 226
You are a database developer on an instance of SQL Server 2008. Your **Project** table contains an **xml** data type column, named **MaterialList**, that contains the material list for each project. The **Project** table is defined as follows:



Project

	Column Name	Data Type	Allow Nulls
🔑	ProjectID	int	☐
	Description	varchar(25)	☑
	StartDate	datetime	☑
	TargetDate	datetime	☑
	Narrative	varchar(MAX)	☑
	SiteID	int	☑
	MaterialList	xml	☑
	CompletionDate	datetime	☑
			☐

The XML in the **MaterialList** column has the following format:

```
<Materials>
 <Item>
  <SKU ID="XK-33" ItemDesc="Corner Brackets" />
  <Quantity>15</Quantity>
  <RequiredDt>03-15-2009</RequiredDt>
 </Item>
 <Item>
  <SKU ID="XV-17" ItemDesc="Metal Shelving" />
  <Quantity>20</Quantity>
  <RequiredDt>03-12-2009</RequiredDt>
 </Item>
</Materials>
```

You have been notified by one of your suppliers that a particular item with a SKU value of 'XS-7' will be backordered. You want to return a list of incomplete projects with material lists that include this item.
Which query should you use?

A.
```
SELECT ProjectID, Description, TargetDate
FROM dbo.ProjectWHERE MaterialList.exist('/Materials/Item/SKU[@ID="XS-7"]')
ANDCompletionDate IS NULL;
```
B.
```
SELECT ProjectID, Description, TargetDate
FROM dbo.ProjectWHERE MaterialList.query('/Materials/Item/SKU[.="XS-7"]')
ANDCompletionDate IS NULL;
```
C.
```
SELECT ProjectID, Description, TargetDate
FROM dbo.ProjectWHERE MaterialList.exist('/Materials/Item/SKU[@ID="XS-7"]') = 1
ANDCompletionDate IS NULL;
```
D.
```
SELECT *, MaterialList.query('Materials/Item/SKU[@ID="XS-7"]')
FROM dbo.Project;
```

Answer: C
Section: (none)

Explanation/Reference:
You can use the **exist()** XML method in the `WHERE` clause of a `SELECT` statement. The **exist()** method accepts a string representing a specific node, and returns a Boolean value indicating if the node exists in the XML. The **exist()** method returns a value of 1 if the specified node exists. In this scenario, you passed it a string to search for the `<SKU>` element with an `ID` attribute value of "XS-7". You also included `CompletionDate ISNULL` in the `WHERE` clause. Therefore, the query will return the **ProjectID**, **Description**, and **TargetDate** for all projects that do not have a
completion date and have a material list that contains the XS-7 item.
You should not use the following query:

```
SELECT ProjectID, Description, TargetDate FROM dbo.ProjectWHERE MaterialList.
exist('/Materials/Item/SKU[@ID="XS-7"]') ANDCompletionDate IS NULL;
```

This query will return an error because the **exist()** method returns a value of 0 or 1. This query will generate the following error:

```
Msg 4145, Level 15, State 1, Line 7An expression of non-boolean type specified in
a context where a condition is expected, near 'AND'.
```

You should not use the following query:

```
SELECT *, MaterialList.query('Materials/Item/SKU[@ID="XS-7"]') FROM dbo.Project;
```

This query will return a row for all projects because no `WHERE` clause was specified. For products with a material list including the XS-7 item, it would return a single `<SKU>` element as follows:

```
<SKU ID="XS-7" ItemDesc="Wall Brackets" />
```

You should not use the following query:

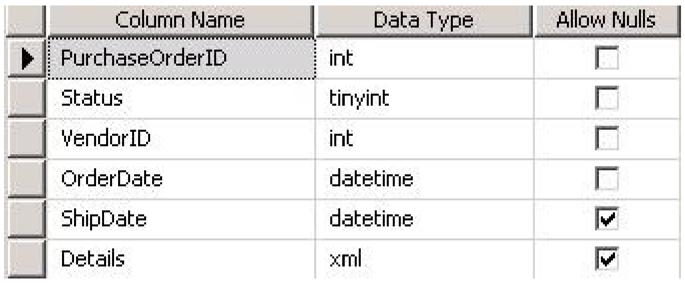
```
SELECT ProjectID, Description, TargetDate FROM dbo.ProjectWHERE MaterialList.
query('/Materials/Item/SKU[.="XS-7"]') ANDCompletionDate IS NULL;
```

The **query()** method is used to query and retrieve XML elements and attributes from an XML instance. The method accepts a string XQuery expression that determines which elements and element attributes are extracted. It will return untyped XML, not a Boolean value that can be referenced in a `WHERE` clause. This query will return the following error:

```
Msg 4145, Level 15, State 1, Line 5An expression of non-boolean type specified in
a context where a condition is expected, near 'AND'.
```

QUESTION 227

You are a database developer on an instance of SQL Server 2008. Your **Prod** database contains a **POMaster** table defined as follows:



Column Name	Data Type	Allow Nulls
PurchaseOrderID	int	☐
Status	tinyint	☐
VendorID	int	☐
OrderDate	datetime	☐
ShipDate	datetime	✓
Details	xml	✓

The **Details** column contains the line items for each purchase order in the following XML format:

```
<POLines PONum="12" VendorName="VirtuArt Corporation">
<POLine>
<InvID>001</InvID>
<Quantity>25</Quantity>
<Cost>125.57</Cost>
<Taxable>1</Taxable>
</POLine>
<POLine>
<InvID>002</InvID>
<Quantity>12</Quantity>
<Cost>29.95</Cost>
<Taxable>0</Taxable>
</POLine>
<POLine>
<InvID>003</InvID>
<Quantity>100</Quantity>
<Cost>2.25</Cost>
<Taxable>1</Taxable>
</POLine>
</POLines>
```

You want to query the **POMaster** table and return the **PurchaseOrderID** and **OrderDate**. You also want the query to return the `PONum` and `VendorName` attributes from the XML stored in the **Details** column for the purchase order.

Which query should you execute?

A.
```
SELECT PurchaseOrderID, OrderDate,
Details.value('(@VendorName) [1]', 'varchar(30)') AS Vendor,
Details.value('(@PONum)[1]', 'int') AS PONumber
FROM POMaster;
```
B.
```
SELECT PurchaseOrderID, OrderDate,
Details.value('(/POLines/@VendorName) [1]', 'varchar(30)') AS Vendor,
Details.value('(/POLines/@PONum)[1]', 'int') AS PONumber
FROM POMaster;
```
C.
```
SELECT PurchaseOrderID, OrderDate,
Details.value('(/POLines/VendorName) [1]', 'varchar(30)') AS Vendor,
Details.value('(/POLines/PONum)[1]', 'int') AS PONumber
FROM POMaster;
```
D.
```
SELECT PurchaseOrderID, OrderDate,
Details.query('(/POLines/@VendorName) [1]', 'varchar(30)') AS Vendor,
Details.query('(/POLines/@PONum)[1]', 'int') AS PONumber
FROM POMaster;
```

Answer: B
Section: (none)

Explanation/Reference:
This query uses the XML **value()** method to return specific attribute values from the XML stored in the **Details** column. The **value()** method accepts
two arguments. The first argument is a string XQuery expression, and the second argument is a string containing a SQL Server data type. The
value() method extracts a value using the XPath expression and returns it as the specified type. In this scenario, you wanted to return the
PurchaseOrderID and **OrderDate** columns from the table and two additional attributes from the XML stored in the **Details** column. The given
`SELECT` statement uses the **value()** method to return a **Vendor** column containing a `varchar(30)` value extracted from the `VendorName` attribute
in the `<POLines>` element of the XML. The statement also uses the **value()** method to return a **PONumber** column containing an `int` value
extracted from the `PONum` attribute in the `<POLines>` element of the XML. The `[1]` at the end of each string expression indicates that the
information is extracted from the first `<POLines>` element. In this scenario, only one `<POLines>` element exists.
You should note that the expression string you specify must return a single value that can be successfully converted to the specified data type. If
you specify an expression that returns multiple values, or an expression that cannot be successfully converted to the specified SQL Server data
type, an error will occur.
You should not execute the following query:
```
SELECT PurchaseOrderID, OrderDate,
Details.value('(@VendorName) [1]', 'varchar(30)') AS Vendor,
Details.value('(@PONum)[1]', 'int') AS PONumber
FROM POMaster;
```
This query will return an error because the expression in the first argument begins with an attribute instead of an element. This is not allowed
because attributes can only be defined within elements. This statement generates the following error:
```
Msg 2390, Level 16, State 1, Line 5
XQuery [POMaster.Details.value()]: Top-level attribute nodes are not supported.
```
You should not execute the following query:
```
SELECT PurchaseOrderID, OrderDate,
Details.value('(/POLines/VendorName) [1]', 'varchar(30)') AS Vendor,
Details.value('(/POLines/PONum)[1]', 'int') AS PONumber
FROM POMaster;
```

This query attempts to return the value of elements named `VendorName` and `PONum`. In this scenario, the XML does not contain elements with
these names. Instead, these are attributes, which should be prefixed with an ampersand (`@`) in the expressions. This query will execute, but will
return a `NULL` value for both **Vendor** and **PONumber** in the result
You should not execute the following query:

```
SELECT PurchaseOrderID, OrderDate,
Details.query('(/POLines/@VendorName) [1]', 'varchar(30)') AS Vendor,
Details.query('(/POLines/@PONum)[1]', 'int') AS PONumber
FROM POMaster;
```

This query returns the following error because the **query()** method only accepts a single argument:

```
Msg 174, Level 15, State 1, Line 4
The query function requires 1 argument(s).
```

Even if the syntax were corrected, using the **query()** method would not be appropriate in this scenario. The **query()** method is used to query and
retrieve XML elements and attributes from an XML instance. The method accepts a string XQuery expression that determines which elements and
attributes are extracted, and returns untyped XML, not scalar values. For example, suppose you used the following query:

```
SELECT PurchaseOrderID, OrderDate,
Details.query('/POLines[@VendorName]') AS Vendor,
Details.query('/POLines[@PONum]') AS PONumber
FROM POMaster;
```

The query would successfully execute, but would contain untyped XML in the **Vendor** and **PONumber** columns of the result.

QUESTION 228
You have a table named SALES with the following columns:

SalesmanID | ProductsSold
1 | 2000
2 | 4000
3 | 5000
4 | 5000
5 | 6000
6 | 6000
7 | 9000
8 | 9000

You have to chose a query that will select all SalesmanIDs that have sold an amount of products within the 3 top ProductsSold values.

A. SELECT TOP(3) WITH TIES
   SalesmanID, ProductsSold
   FROM Sales
   order by ProductsSold
B. SELECT TOP(30) PERCENT
   WITH TIES
   SalesmanID, ProductsSold
   FROM Sales
   order by ProductsSold

C. SELECT SalesmanID
   From Sales
   where ProductsSold in (Select TOP 3 ProductsSold
   From Sales)

D. Select TOP (3) ProductsSold, SalesmanID
   From Sales

Answer: A
Section: (none)

Explanation/Reference:
http://msdn.microsoft.com/en-us/library/ms189463.aspx

SYNTAX:
[
    TOP (expression) [PERCENT]
    [WITH TIES]
]

WITH TIES

Specifies that additional rows be returned from the base result set with the same value in the ORDER BY columns appearing as the last of the TOP n (PERCENT) rows. TOP...WITH TIES can be specified only in SELECT statements, and only if an ORDER BY clause is specified.

QUESTION 229
You manage a database on an instance of SQL Server 2008 for a large training company.
You have the following Transact-SQL:
```
DECLARE @xmldoc xmlSET @xmldoc='<EventData> <Event ID="C-012" EventDesc="Using
Microsoft Word 2007" EventLoc="Southeast Campus"><Coordinator>Beth Jackson</
Coordinator><AvailSeats>35</AvailSeats><Enrolled>10</Enrolled></Event><Event
ID="C-057" EventDesc="SQL Server 2008 Basics" EventLoc="Main
Campus"><Coordinator>Ross Moore</Coordinator><AvailSeats>30</
AvailSeats><Enrolled>15</Enrolled></Event><Event ID="M-023" EventDesc="Time
Management 101" EventLoc="Adair Center"><Coordinator>Helen Lewis</
Coordinator><AvailSeats>45</AvailSeats><Enrolled>30</Enrolled></Event></
EventData>'
```
You want to extract data from the XML document and insert it into a new table named **EventList**. The new table should contain a row for each event that includes the following columns:
EventID - The `ID` attribute value for the event
Description - The `EventDesc` attribute value for the event
Location - The `EventLoc` attribute value for the event
Which Transact-SQL statement should you use?

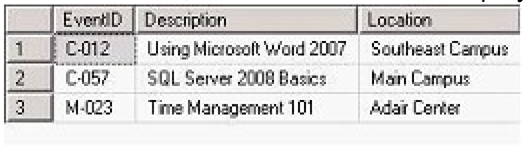
A.
```
SELECT
x.Event.value('@ID[1]', 'varchar(10)') AS EventID,x.Event.value('@EventDesc
[1]', 'varchar(35)') AS Description,x.Event.value('@EventLoc[1]', 'varchar
(25)') AS LocationINTO EventList FROM @xmldoc.nodes('//EventData/Event') AS x
(Event);
```
B.
```
SELECT
@xmldoc.query('/EventData/Event[@ID]') AS EventID,@xmldoc.query('/EventData/
Event[@EventDesc]') AS Description,@xmldoc.query('/EventData/Event[@EventLoc]')
AS LocationINTO EventList;
```

C.
```
SELECT
@xmldoc.query('/EventData/Event/@ID') AS EventID,@xmldoc.query('/EventData/
Event/@EventDesc') AS Description,@xmldoc.query('/EventData/Event/@EventLoc')
AS LocationINTO EventList;
```

D.
```
SELECT
@xmldoc.value('/EventData/Event/@ID', 'varchar(10)') AS EventID,@xmldoc.value
('/EventData/Event/@EventDesc', 'varchar(35)') AS Description,@xmldoc.value('/
EventData/Event/@EventLoc', 'varchar(25)') AS LocationINTO EventList;
```

Answer: A
Section: (none)

Explanation/Reference:
In this scenario, the query uses the **value()** XML method to extract values of specific elements and attributes. The **value()** method accepts two arguments. The first argument is a string XQuery expression, and the second argument is a string containing a SQL Server data type. The **value()** method extracts a single value using the XPath expression and returns it as the specified type. In this scenario, the statement uses the **value()** method to return an **EventID** column containing a `varchar(10)` value extracted from the `ID` attribute in the `<Event>` element of the XML. The statement also uses the **value()** method to return a **Description** column containing a `varchar(35)` value extracted from the `EventDesc` attribute in the `<Event>` element of the XML. The `[1]` at the end of each string expression indicates that the information is extracted from a single element. The **value()** method must return a single value. However, in this scenario, the XML contains multiple events. Therefore, the correct query includes the **nodes()** XML method in the `FROM` clause of the query. The **nodes()** method accepts an XQuery string and returns all of the specified nodes as a result set. In this scenario, the **nodes()** method returns a result set that contains one row for each event. Each row contains the corresponding XML for the event. Then, in the `SELECT` list, the table alias defined is used with the **value()** method to return the details for each specific event. You could also use the **nodes()** method to separate XML. In this scenario, the statement would display results similar to the following:



	EventID	Description	Location
1	C-012	Using Microsoft Word 2007	Southeast Campus
2	C-057	SQL Server 2008 Basics	Main Campus
3	M-023	Time Management 101	Adair Center

You should not use the following query:
```
SELECT @xmldoc.value('/EventData/Event/@ID', 'varchar(10)') AS EventID,@xmldoc.
value('/EventData/Event/@EventDesc', 'varchar(35)') AS Description,@xmldoc.value
('/EventData/Event/@EventLoc', 'varchar(25)') AS LocationINTO EventList;
```
The **value()** method must return a single value. This query omitted the `[1]` at the end of each expression that indicates a single value is returned. If
you specify an expression that returns multiple values, or an expression that cannot be successfully converted to the specified SQL Server data type, an error occurs. This query will return the following error:
```
Msg 2389, Level 16, State 1, Line 25XQuery [value()]: 'value()' requires a
singleton (or empty sequence), found operand of type 'xdt:untypedAtomic *'
```
You should not use the following query:
```
SELECT @xmldoc.query('/EventData/Event[@ID]') AS EventID,@xmldoc.query('/
EventData/Event[@EventDesc]') AS Description,@xmldoc.query('/EventData/Event
[@EventLoc]') AS LocationINTO EventList;
```
This query uses the **query()** method. The **query()** method is used to query and retrieve XML elements and attributes from an XML instance. The method accepts a string XQuery expression that determines which elements and element attributes are extracted. This query will execute successfully, but will not produce the desired results. This query will insert one row into the **EventList** table that contains three columns, **EventID**, **Description**, and **Location**. However, each column will contain the following XML:
```
<Event ID="C-012" EventDesc="Using Microsoft Word 2007" EventLoc="Southeast
Campus"><Coordinator>Beth Jackson</Coordinator><AvailSeats>35</
AvailSeats><Enrolled>10</Enrolled></Event><Event ID="C-057" EventDesc="SQL Server
2008 Basics" EventLoc="Main Campus"><Coordinator>Ross Moore</
Coordinator><AvailSeats>30</AvailSeats><Enrolled>15</Enrolled></Event><Event
ID="M-023" EventDesc="Time Management 101" EventLoc="Adair
```

```
Center"><Coordinator>Helen Lewis</Coordinator><AvailSeats>45</
AvailSeats><Enrolled>30</Enrolled></Event>
```

You should not use the following query:

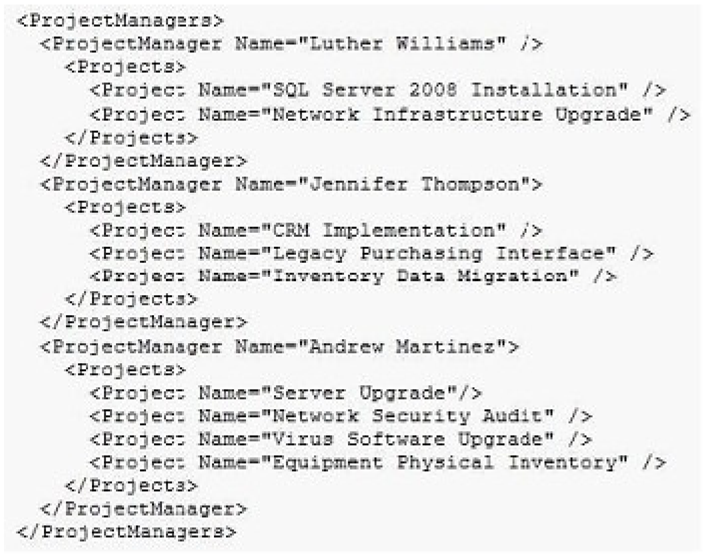
```
SELECT @xmldoc.query('/EventData/Event/@ID') AS EventID,@xmldoc.query('/
EventData/Event/@EventDesc') AS Description,@xmldoc.query('/EventData/Event/
@EventLoc') AS LocationINTO EventList;
```

This statement generates the following syntax error because the XQuery expression is not valid:

```
Msg 2396, Level 16, State 1, Line 25XQuery [query()]: Attribute may not appear
outside of an element
```

QUESTION 230

You are a database developer on an instance of SQL Server 2008. You receive an XML file containing information on project managers and their current projects. The format of the XML file you received is shown as follows:



```
<ProjectManagers>
  <ProjectManager Name="Luther Williams" />
    <Projects>
      <Project Name="SQL Server 2008 Installation" />
      <Project Name="Network Infrastructure Upgrade" />
    </Projects>
  </ProjectManager>
  <ProjectManager Name="Jennifer Thompson">
    <Projects>
      <Project Name="CRM Implementation" />
      <Project Name="Legacy Purchasing Interface" />
      <Project Name="Inventory Data Migration" />
    </Projects>
  </ProjectManager>
  <ProjectManager Name="Andrew Martinez">
    <Projects>
      <Project Name="Server Upgrade"/>
      <Project Name="Network Security Audit" />
      <Project Name="Virus Software Upgrade" />
      <Project Name="Equipment Physical Inventory" />
    </Projects>
  </ProjectManager>
</ProjectManagers>
```

From the file received, you need to generate another XML file that contains each project manager and the number of projects for which the manager is responsible. You want the list of project managers to be ranked from highest to lowest based on the number of projects they currently manage. You want to generate the XML output in the following format:

```
<ProjectManagers>
 <ProjectManager Name="Andrew Martinez">
  <ProjectCount>4</ProjectCount>
 </ProjectManager>
 <ProjectManager Name="Jennifer Thompson">
  <ProjectCount>3</ProjectCount>
 </ProjectManager>
 <ProjectManager Name="Luther Williams">
  <ProjectCount>2</ProjectCount>
 </ProjectManager>
</ProjectManagers>
```

You declare an **xml** variable named `@xml` and populate it with the incoming data.
Which query will produce the desired result?

A.
```
SELECT @xml.query(COUNT(x.Projects.value('@Name[1]', 'varchar(30)')) AS
ProjectCount FROM @xml.nodes('//ProjectManagers/ProjectManager') AS x(Projects)
GROUP BY x.Projects.value('ProjectManager[1]','varchar(30)')ORDER BY COUNT(x.
Projects.value('@Name[1]', 'varchar(30)')) DESC;
```

B.
```
SELECT @xml.query('<ProjectManagers>{for $ProjectManager in /ProjectManagers/
ProjectManagerlet $count := count($ProjectManager/Projects/Project)order by
$count descending
return <ProjectManager>{$ProjectManager/@Name}{$ProjectManager/count}
<ProjectCount>{$count}</ProjectCount></ProjectManager>}</ProjectManagers>');
```
C.
```
SELECT x.Projects.value('ProjectManager[1]','varchar(30)') As ProjectManager,
'<ProjectManagers>{for $ProjectManager in /ProjectManagers/ProjectManagerlet
$count := count($ProjectManager/Projects/Project)order by $count descending
return <ProjectManager>{$ProjectManager/@Name}{$ProjectManager/count}
<ProjectCount>{$count}</ProjectCount></ProjectManager>}</ProjectManagers>');
```
D.
```
SELECT @xml.query(
COUNT(x.Projects.value('@Name[1]', 'varchar(30)')) AS ProjectCount FROM @xml.
nodes('//ProjectManagers/ProjectManager') AS x(Projects)GROUP BY x.Projects.
value('ProjectManager[1]','varchar(30)')ORDER BY COUNT(x.Projects.value('@Name
[1]', 'varchar(30)')) DESC FOR XML AUTO;
```

Answer: B
Section: (none)

Explanation/Reference:
This query uses the **query()** method and includes a FLWOR expression. The **query()** method is used to query and retrieve XML elements and attributes from an XML instance. The method accepts a string XQuery expression that determines which elements and attributes are extracted, and returns untyped XML. FLWOR expressions allow you to use XQuery to process XML using a construct similar to a SQL `SELECT` statement. The term FLWOR is derived from the clauses used in the expression, namely **for**, **let**, **where**, **order by**, and **return**. The clauses of a FLWOR expression are defined as follows:
for: Defines the elements processed by the expression. This required clause is similar to the `FROM` clause of a `SELECT` statement.
let: Defines a variable that can be used in the expression. This optional clause can include calls to XQuery functions. This clause was not allowed in previous SQL Server versions, but is supported in SQL Server 2008.

where: Filters the result returned. This optional clause is similar to the `WHERE` clause of a `SELECT` statement.
• **order by**: Orders the result returned. This optional clause is similar to the `ORDER BY` clause of a `SELECT` statement.
return: Returns the result. This clause specifies the XML nodes that will be returned.

Variables in a FLWOR expression are prefixed with a dollar sign (`$`). The FLWOR expression iterates over the XML nodes specified by the clause and returns the specified result. In this scenario, the FLOWR expression iterates over all `<ProjectManager>` elements in the XML, which are subelements of each `<ProjectManagers>` element. The **let** clause defines a variable named **$count** and calls the **count** XQuery function to count the number of projects for the project manager. The **order by** clause uses the **$count** variable value to sort the result in descending order by the number of projects, and the **return** clause specifies the elements to be returned.
The two queries that include a `GROUP BY` clause will generate the following error because XML methods are not valid in a `GROUP BY` clause:
```
Msg 4148, Level 16, State 1, Line 30XML methods are not allowed in a GROUP BY
clause.
```
The following query will not produce the desired result:
```
SELECT @xml.query('<ProjectManagers>{for $ProjectManager in /ProjectManagers/
ProjectManagerlet $count := count($ProjectManager/Projects/Project)order by
$count descendingreturn <ProjectManager>{$ProjectManager/@Name}{$ProjectManager/
count}<ProjectCount>count</ProjectCount></ProjectManager>}</ProjectManagers>');
```
This query specifies a literal value within the `<ProjectCount>` element in the **return** clause. This query will generate output in the following format:
```
<ProjectManagers><ProjectManager Name="Andrew Martinez"><ProjectCount>count</
ProjectCount></ProjectManager><ProjectManager Name="Jennifer
Thompson"><ProjectCount>count</ProjectCount></ProjectManager><ProjectManager
```

```
Name="Luther Williams"><ProjectCount>count</ProjectCount></ProjectManager></
ProjectManagers>
```

QUESTION 231
You are a database developer. You plan to design a database solution by using SQL Server 2008. A stored procedure in a database uses a transaction to retrieve data from a table and produces aggregations.
You must design a solution that meets the following requirements:
·
Update operations cannot be performed on the retrieved data while the stored procedure is being executed.
·
Insert operations in the table can be performed while the stored procedure is being executed. You need to ensure that the solution meets the requirements.
What isolation level should you use?

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

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 232
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. There's a stored procedure in a database. This stored procedure retrieve data from a table and produces aggregations by using a transaction.
Now you have to design a solution and it must satisfy the requirements below:
When the stored procedure is being executed, insert operations in the table can be performed. When the stored procedure is being executed, update operations cannot be performed on the retrieved data.
You must make sure that the requirements can be met. So what isolation level should you use?

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

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 233
You manage a SQL Server 2008 database instance. You want to schedule specific administrative tasks to run each weekend at an off-peak time. If any of the scheduled administrative tasks fails, you want to notify the database administrator on call.
Which SQL Server component should you use?

A. Serice Broker

B. Database Mail
C. SQL Server Agent
D. Distributed Transaction Coordinator (DTC)

Answer: C
Section: (none)

Explanation/Reference:
You should use SQL Server Agent. SQL Server Agent is a service that runs in the background and is used to execute jobs, monitor the SQL server, and process alerts. A job consisting of one or more job steps can be run on demand, when a specific event occurs, or on a defined schedule. SQL Server Agent is often used to perform administrative tasks at predefined times. SQL Server Agent also monitors the SQL server and can fire defined alerts when a given error or event occurs. For example, in this scenario, you could create a job with job steps to perform the desired administrative tasks and schedule the job to run each weekend at an off-peak time. Then, you could create an alert that would notify the on-call database administrator if the job fails.
You should not use Service Broker. Service Broker is part of the database engine that provides for queuing and messaging. It coordinates sending and receiving messages, provides for prioritizing conversations, and can use automatic activation to automatically process messages when they are received.
You should not use Database Mail. You can code applications to send e-mail messages via Database Mail. Messages can be sent using the **sp_send_dbmail** system stored procedure. However, in this scenario, you would also need to use SQL Server Agent to create an alert that would fire in response to failed maintenance tasks. Database Mail sends e-mail via the Simple Mail Transfer Protocol (SMTP) and does not require that an extended MAPI client be installed on the server. Database Mail allows for query results to be included in the e-mail or as an attachment, can limit attachment file sizes and types, and is much more secure than SQL Server Mail.
You should not use Distributed Transaction Coordinator (DTC). Microsoft Distributed Transaction Coordinator (MS DTC) is used to provide for transactional processing across multiple SQL Server instances. This allows you to ensure that modifications on the first instance are only committed if the actions performed on the second instance are successful. If the actions on the second instance are unsuccessful, you could roll back the modifications performed on the original instance.

QUESTION 234
You have two different application sessions. What solution should you implement to share transactions and lock between them to avoid any conflicts

A. SET REMOTE_PROC_TRANSACTIONS ON
B. Use BEGIN DISTRIBUTED TRANSACTION in the stored procedure.
C. Use Bound sessions
D. Use unBound sessions

Answer: C
Section: (none)

Explanation/Reference:
http://msdn.microsoft.com/en-us/library/ms177480.aspx

Bound sessions ease the coordination of actions across multiple sessions on the same server. Bound sessions allow two or more sessions to share the same transaction and locks, and can work on the same data without lock conflicts. Bound sessions can be created from multiple sessions within the same application or from multiple applications with separate sessions.

QUESTION 235
You have a database that resides on an instance of SQL Server 2008. You are creating a stored procedure that will update a critical table. The
stored procedure reads data in the **Master** table, performs aggregation calculations on the table's data, and populates a `table` variable with the result.
You must ensure that the variable always contains the most up-to-date information from the **Master** table, and you want to lock the table exclusively while the query runs.
Which action should you take?

A. Set the `ALLOW_SNAPSHOT_ISOLATION` database option to `ON`.
B. Perform the aggregate calculations within a transaction, and set the transaction's isolation level to `REPEATABLE READ`.
C. Include the `READPAST` query hint in the query containing the aggregate calculations.
D. Add the `TABLOCKX` hint to the query containing the aggregate calculations.

Answer: D
Section: (none)

Explanation/Reference:
In this scenario, you wanted to lock the **Master** table exclusively for a given query. To do so, you can use the `TABLOCKX` query hint.
You should not set the `ALLOW_SNAPSHOT_ISOLATION` database option to `ON`. The `ALLOW_SNAPSHOT_ISOLATION` database option controls whether transactions with a `SNAPSHOT` isolation level are allowed. A value of `ON` indicates transactions with a `SNAPSHOT` isolation level are allowed.
You should not include the `READPAST` query hint in the query containing the aggregate calculations because this would not ensure the query always received the most up-to-date data. The `READPAST` query hint ignores any current locks and reads past those rows. Any locked rows would not be included in your aggregations.
You should not perform the aggregate calculations within a transaction, and set the transaction's isolation level to `REPEATABLE READ`. In this scenario, you only wanted to lock the table during the duration of a query. Therefore, using a table hint would be sufficient.

QUESTION 236
You are a database developer for a large automobile manufacturer. Users accessing a large partitioned table in the **Prod** database complain that their queries take a long time to complete.
You run a SQL Server Profiler trace including the **Lock:Escalation** event, and notice that an excessive number lock escalations are occurring for the table. You would like to force SQL Server to take partition-level locks when possible to see if this improves query response time.
Which action should you take?

A. Set the `READ_COMMITTED_SNAPSHOT` database option to `OFF`.
B. Use **sp_configure** to set the **locks** option to a lower value.
C. Use trace flag 1211 to configure lock escalation.
D. Issue an `ALTER TABLE` statement to set the table's `LOCK_ESCALATION` to `AUTO`.

Answer: D
Section: (none)

Explanation/Reference:
To force SQL Server to use partition-level locks for a partitioned table, you should issue an `ALTER TABLE` statement to set the table's `LOCK_ESCALATION` to `AUTO`. By default, SQL Server will escalate locks to the table level when required. When an excessive number of row-level locks are encountered, SQL Server will escalate the lock to a higher level. This causes the entire table to be locked, which is usually sufficient and

reduces memory consumption. However, with partitioned tables, you may not want to lock the entire table, but only a single partition, such as the partition containing rows being modified. This would allow queries against other partitions in the table to execute successfully without being blocked. You can use the `LOCK_ESCALATION` setting for a table to control how SQL Server will escalate locks. The `LOCK_ESCALATION` setting can have one of the following values:
`TABLE`: Lock escalation is performed to the table level, which is the default.
`AUTO`: Lock escalation is performed to the table level for non-partitioned tables, and to the partition level for partitioned tables.
`DISABLE`: Lock escalation is disabled, and lock escalation does not typically occur. SQL Server only escalates locks in specific situations where it is absolutely required to ensure data integrity.

In this scenario, you could set the `LOCK_ESCALATION` setting to `AUTO` for the table, and partition-level lock escalation would be used.
You should not set the `READ_COMMITTED_SNAPSHOT` database option to `OFF`. The `READ_COMMITTED_SNAPSHOT` database option controls how transactions with the `READ COMMITTED` isolation level are handled, which would not help in this scenario. When the `READ_COMMITTED_SNAPSHOT` database option is set to `ON`, transactions with the `READ COMMITTED` isolation level use row versioning instead of locking.
You should not use **sp_configure** to set the **locks** option to a lower value. The **locks** configuration option is used to limit the number of locks at the server level. You can adjust this setting to minimize the amount of memory consumed if necessary. However, the default value of 0 allows SQL Server to dynamically manage the locks as needed.
You should not use trace flag 1211 to configure lock escalation because this would disable lock escalation for the entire instance. With this flag set, SQL Server will not escalate row or page-level locks for any tables in the database.

QUESTION 237
You are a database administrator on an instance of SQL Server 2008. You want to create a table named **InvTransaction** that contains the following
information:
- **TranDate** - A transaction date and time
- **ProductID** - A product identifier
- **TranCOA** - A transaction code corresponding to the company's chart of accounts
- **LocationID** - A warehouse location identifier
- **BinID** - A bin location identifier
- **Quantity** - A quantity
- **TranType** - A transaction type
- **TranDetails** - A detailed transaction description

The data will be imported from a legacy system, and the majority of transactions will not have values to be stored in the **LocationID** and **BinID**
columns. You want to minimize storage requirements for the table.
What should you do?

A. Set the **text in row** table option to `OFF`.
B. Use FILESTREAM storage.
C. Implement the **LocationID** and **BinID** columns as sparse columns.
D. Create a column set.

Answer: C
Section: (none)

Explanation/Reference:
When creating a table, sparse columns can to optimize the storage
of `NULL` values. In this scenario, the majority of rows that will be imported will not have **LocationID** and

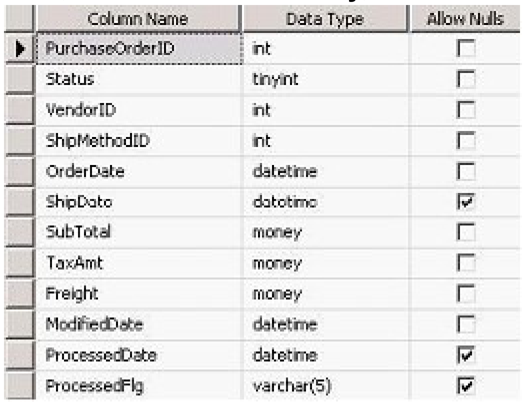
BinID values. You can minimize storage
requirements by defining these columns as sparse columns. For example, you could use the following statement to create the **InvTransaction** table
with **LocationID** and **BinID** as sparse columns:

```
CREATE TABLE InvTransaction (
TranDate datetime,
ProductID int,
TranCOA int,
LocationID int SPARSE NULL,
BinID int SPARSE NULL,
Quantity smallint,
TranType char(2),
TranDetails varchar(max));
```

You should not create a column set. Column sets can be used with a table that contains sparse columns to be able to return all sparse column
values as XML, but would not be required in this scenario because you are not returning data in sparse columns.
You should not use FILESTREAM storage. FILESTREAM storage is implemented to store large binary objects as files on the file system, such as
image or video files, and manage them using Transact-SQL. FILESTREAM data can also be accessed using Win32 APIs. FILESTREAM storage is
tightly integrated with most database functionality, including backup and recovery. When you take a database backup, FILESTREAM storage is also
backed up unless you override this functionality by performing a partial backup. To create a table that can store FILESTREAM data, you create a
table that contains a column of the **varbinary(max)** data type and include the `FILESTREAM` attribute.
You should not set the **text in row** table option to `OFF`. The **text in row** option is used to control whether **text**, **ntext**, or **image** columns are stored
in the data row. You can also use the **large value types out of row** option to specify whether large value types, such as **varchar(max)**, **nvarchar**
(max), and **varbinary(max)**, are stored in the data row.

QUESTION 238

You have a **Prod** database on an instance of SQL Server 2008. Your **Prod** database contains the following table, **PurchaseOrderHistory**, which contains many rows:



Column Name	Data Type	Allow Nulls
PurchaseOrderID	int	☐
Status	tinyint	☐
VendorID	int	☐
ShipMethodID	int	☐
OrderDate	datetime	☐
ShipDate	datetime	✓
SubTotal	money	☐
TaxAmt	money	☐
Freight	money	☐
ModifiedDate	datetime	☐
ProcessedDate	datetime	✓
ProcessedFlg	varchar(5)	✓

You have two stored procedures that have the following characteristics:
StoredProc1 has a default isolation level and issues multiple DML statements that perform modifications to the **PurchaseOrderHistory** table.
StoredProc2 has a default isolation level and issues a `SELECT` query to return all rows in the **PurchaseOrderHistory** table.

You want to ensure that **StoredProc2** uses row versioning and retrieves the last committed version of each row. Which action should you take?

A. Set the transaction isolation level of **StoredProc1** to `SERIALIZABLE`.
B. Include the `NOLOCK` table hint for the query in **StoredProc2**.
C. Set the `ALLOW_SNAPSHOT_ISOLATION` database option to `OFF`.
D. Set the `READ_COMMITTED_SNAPSHOT` database option to `ON`.

Answer:
Section: (none)

Explanation/Reference:
This setting allows SQL Server to use row versioning instead of locking for all transactions with an isolation level of `READ COMMITTED`. This provides read consistency at the statement level using row versions stored in the **tempdb** database. Each time a DML statement within the transaction executes, a new snapshot of the row versions is stored in the **tempdb** database and is used if other transactions access the modified rows. Using row versioning can significantly increase the size of the **tempdb**database. Therefore, you should closely analyze your requirements before using row versioning. You might choose to use statement-level row versioning when you are running queries against data that must be accurate as of the time that other queries modifying the data started. For transactions with other isolation levels, you might consider using a **rowversion** column to implement row-versioning capability manually.
You should not set the transaction isolation level of **StoredProc1** to `SERIALIZABLE`. The `SERIALIZABLE` transaction isolation level completely isolates transactions from each other, and is the most restrictive isolation level. If you used a `SERIALIZABLE` transaction isolation level, **StoredProc2** could not access the data until **StoredProc1** completed. The `SERIALIZABLE` transaction isolation level increases the possibility of blocking and should be avoided unless it is absolutely necessary. In this scenario, there was no requirement to modify **StoredProc1**'s transaction isolation level.
You should not include the `NOLOCK` table hint for the query in **StoredProc2**. The `NOLOCK` table hint is specified for a single table in a query to ignore any locked rows in the table. Using this hint in a query will allow the query to execute, even if locks exist, but the query may return uncommitted rows.
You should not set the `ALLOW_SNAPSHOT_ISOLATION` database option to `OFF`. The `ALLOW_SNAPSHOT_ISOLATION` database option controls whether transactions with a transaction isolation level of `SNAPSHOT` are allowed. In this scenario, both transactions used the default transaction isolation level, which is `READ COMMITTED`. Therefore, modifying this option would have no effect. When this option is turned on, transactions with the `SNAPSHOT` isolation level use row versioning at the transaction level, rather than at the statement level.

QUESTION 239
You are database developer on an instance of SQL Server 2008. The development team has recently created and compiled an assembly using
a .NET Framework language. The assembly includes a method that requires access to the file system. You want to allow a user to access the method directly from Transact-SQL.
Which action should you take?

A. Set the `TRUSTWORTHY` database property to `OFF`.
B. Grant the user access to the stored procedure using a database role that has elevated privileges.
C. Grant the user the **db_accessadmin** fixed database role.
D. Register the assembly with the `EXTERNAL_ACCESS` permission.

Answer: D
Section: (none)

Explanation/Reference:
When you register a .NET assembly using the `CREATE ASSEMBLY` statement, you can specify a permission set that identifies the actions that can be performed. You may specify one of the following permission sets

depending on which actions should be allowed:
`SAFE`: Provides access to the current database, but does not allow the function to access external files, the registry, or environment variables, or to call unmanaged code.
`EXTERNAL_ACCESS`: Provides access to the current database and external resources, including the file system, registry, environment variables, and unmanaged code.
`UNSAFE`: Provides full trust access, and the CLR does not perform any permission checks.

The default permission set is `SAFE`. In this scenario, the assembly needs to access the file system. Therefore, you should specify a `PERMISSION_SET` of `EXTERNAL_ACCESS`. For example, you might use the following statement to register an assembly that is allowed to access the file system:

```
CREATE ASSEMBLY MyAssemblyFROM 'C:\Test.dll' WITH PERMISSION_SET =
EXTERNAL_ACCESS;
```

After you register the assembly, you can use a `CREATE PROCEDURE` statement with the `EXTERNAL NAME` clause to specify the name of the method the user needs to access, and the user will be able to access the CLR stored procedure directly from Transact-SQL.
You should not set the `TRUSTWORTHY` database property to `OFF`. The `TRUSTWORTHY` database property should be set to `ON` to allow the assembly to access external resources, such as the file system.
You should not grant the user access to the stored procedure using a database role that has elevated privileges. There is no need to create a special database role to provide access to the file system. You can accomplish this using the permission set of the assembly.
You should not grant the user the **db_accessadmin** fixed database role because this will not allow the function to access the file system. This would also allow the user to perform actions that are not desired, such as creating a schema or adding or removing users. Granting fixed database roles should be avoided.

QUESTION 240
You are a database developer. You develop solutions by using SQL Server 2008 in an enterprise environment.You plan to create a stored procedure that queries a sales table and produces forecast data. You do not have administrative permissions, and you are not the owner of the database. You have permissions to create stored procedures. Users will only have permissions to execute your stored procedures.
You need to ensure that users can execute the stored procedures.
What should you do?

A. Set the TRUSTWORTHY property of the database to ON.
B. Include an EXECUTE AS OWNER clause when you create each stored procedure.
C. Include an EXECUTE AS CALLER clause when you create each stored procedure.
D. Include a SETUSER statement before you query the sales table in each stored procedure.

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 241
You are a database developer on an instance of SQL Server 2008. You want to write Transact-SQL code to modify data in two separate tables, the **Inventory** table and the **InvHistory** table. First, the **Inventory** table should be updated. If the update to the **Inventory** table is successful, then a row should be inserted into the **InvHistory** table.
If either of the modifications fails, both should be rolled back.
What should you do?

A. Set the `XACT_ABORT` option to `OFF` and enclose the code within a single explicit transaction.

B. Include the code for both DML operations in a `TRY` block, and include a `CATCH` block that includes a `ROLLBACK TRANSACTION` statement.
C. Issue the `UPDATE` and `INSERT` statements, then check `@@ERROR` and roll back the transaction if `@@ERROR` returns a non-zero value.
D. Set the `XACT_ABORT` option to `ON` and enclose the code within a single explicit transaction.

Answer: D
Section: (none)

Explanation/Reference:
Transactions can be specified explicitly, implicitly, or automatically. In autocommit mode, which is the default, each Transact-SQL statement is treated as a separate transaction, and each statement is automatically committed when it successfully executes. To ensure that both modifications either succeed or fail, you should include them within an explicit transaction.
The `XACT_ABORT` option controls how SQL Server handles transactions when a run-time error occurs. When the `SET XACT_ABORT` option is set to `ON`, and a Transact-SQL statement raises an error at run time, the entire transaction is terminated and rolled back. When the `XACT_ABORT` option is set to `OFF`, only the Transact-SQL statement that raised the error is rolled back. The remaining statements in the transaction will be executed. The default value of the `XACT_ABORT` option is `OFF`. In this scenario, you wanted to ensure that either both statements execute successfully or both are rolled back. Setting `XACT_ABORT` to `ON` before the transaction executes will ensure that if either statement generates an error, both statements will be rolled back. For example, in this scenario, you might use the following code:

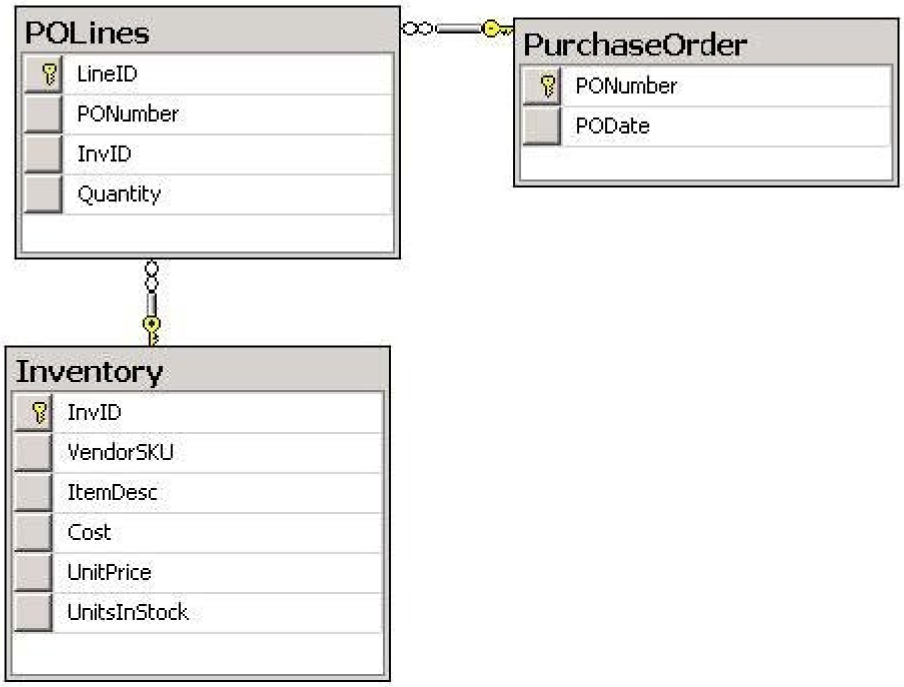
```
BEGIN TRANSACTION
UPDATE Inventory SET UnitsInStock = 0 WHERE InvID = 4;INSERT INTO InvHistory
(InvID, ModifiedDate)VALUES (4, GETDATE());
COMMIT TRANSACTION
```

If either DML statement fails, the entire transaction would be rolled back.
You can also use the `ROLLBACK` statement and other procedural constructs within a block of Transact-SQL code to control when transactions are committed or rolled back. You might want to return a custom error message or perform other actions when specific statements are unsuccessful. To accomplish this, you would need to include additional error-handling code. For example, the following code would accomplish the same result, but you could include custom error processing within each `BEGIN...END` block:

```
BEGIN TRANSACTION UPDATE Inventory SET UnitsInStock = 0 WHERE InvID = 4;IF
@@ERROR <> 0BEGIN ROLLBACK RETURN END INSERT INTO InvHistory(InvID, ModifiedDate)
VALUES (4, GETDATE());IF @@ERROR <> 0BEGIN ROLLBACK RETURN END ELSE COMMIT
```

You should not set the `XACT_ABORT` option to `OFF` and enclose the code within a single explicit transaction. With this setting, if one of the statements failed, only the failed statement would be rolled back, rather than the entire transaction.
You should not issue the `UPDATE` and `INSERT` statements, then check `@@ERROR` and roll back the transaction if `@@ERROR` returns a non-zero value. The `@@ERROR` function returns a non-zero value, but only for the most recently executed statement. Therefore, you would have to check the value of `@@ERROR` after each DML statement.
You should not include the code for both DML operations in a `TRY` block, and include a `CATCH` block. Although you can use a `TRY...CATCH` construct to customize error handling in a transaction, simply including the DML within a `TRY` block and including a `CATCH` block is not sufficient. You would need to include the appropriate code to roll back the transaction in the `CATCH` block if an error occurred.

QUESTION 242
You are a database developer on an instance of SQL Server 2008. Your **Prod** database contains the following tables:



You have a stored procedure, **AddPurchaseOrder**, which accepts a table-valued input parameter containing the purchase order details, begins a transaction, and inserts the appropriate rows into the**PurchaseOrder** and **POLines** tables.
The **AddPurchaseOrder** stored procedure then calls the **UpdInventory** stored procedure to update inventory quantities for the products sold on the purchase order. The **UpdInventory** stored procedure is defined as follows:

```
CREATE PROCEDURE UpdInventory(@InvID int, @qty int)AS
BEGIN TRANSACTION
UPDATE InventorySET UnitsInStock = UnitsInStock - @qty WHERE InvID = @InvID;
INSERT INTO InvQtyHistory(InvID, Adj, Description)VALUES (@InvID, @qty,
'Inventory adjustment for purchase order');
COMMIT TRANSACTION
```

The **Inventory** table contains a `CHECK` constraint on the **UnitsInStock** column that prevents it from being updated to a negative value. You want to ensure that if the **UpdInventory** stored procedure attempts to update the **UnitsInStock** table with a negative value, then no inserts or updates to any of the tables will be committed.
Which action should you take?

A. Set the `XACT_ABORT` option to `OFF` at the beginning of the **AddPurchaseOrder** stored procedure.
B. Set the `XACT_ABORT` option to `ON` at the beginning of the **AddPurchaseOrder** stored procedure.
C. Set `the XACT_ABORT` option to `ON` at the beginning of the **UpdInventory** stored procedure.
D. Include the code for both stored procedures in a single stored procedure that includes a `TRY...CATCH` block.

Answer: B
Section: (none)

Explanation/Reference:
The `XACT_ABORT` option controls how SQL Server handles transactions when a run-time error occurs. When the `XACT_ABORT` option is set to `ON`, and a Transact-SQL statement raises an error at run time, the entire transaction is terminated and rolled back. When the `XACT_ABORT` option is set to `OFF`, only the Transact-SQL statement that raised the error is rolled back. The remaining statements in the transaction will be executed.

The default value of the `XACT_ABORT` option is `OFF`.
In this scenario, you also have nested transactions. The **AddPurchaseOrder** stored procedure begins a transaction and then calls the **UpdInventory** stored procedure, which begins another transaction. When transactions are nested, the outer transaction controls whether or not both transactions are committed or rolled back. Therefore, to ensure that all modifications take place or are rolled back, you can set the `XACT_ABORT` option to `ON` before beginning the outer transaction, namely the transaction started in **AddPurchaseOrder**. If an error occurs in **AddPurchaseOrder**, including an invalid update in **UpdInventory**, all statements in both transactions will be rolled back.
You should not set the `XACT_ABORT` option to `OFF` at the beginning of the **AddPurchaseOrder** stored procedure. With this setting, each statement within the **AddPurchaseOrder** stored procedure would be considered individually. Some of the modifications might be committed, while others might not.
You should not set `the XACT_ABORT` option to `ON` at the beginning of the **UpdInventory** stored procedure. This would control how modifications are committed within the **UpdInventory** stored procedure, but not in the outer procedure.
You should not include the code for both stored procedures in a single stored procedure that includes a `TRY...CATCH` block. Although you can use a `TRY...CATCH` block to perform error processing, the **UpdInventory** stored procedure might be called from other Transact-SQL code. Therefore, this would not be the best choice, because it could affect other code.

QUESTION 243
You are a database developer on an instance of SQL Server 2008. Daily, you receive XML documents containing detailed order data from several business partners, which use different database platforms.
You need to be able to process the incoming data to update tables in your existing **Sales** database. You will create stored procedures to perform these updates.
You also need to extract portions of the XML documents to construct other XML documents that will be submitted as purchase orders to several of your vendors. You need the queries that perform the extraction to run as quickly as possible. These generated XML documents must conform to a standard vendor format, defined in an XSD.

Which action should you take?

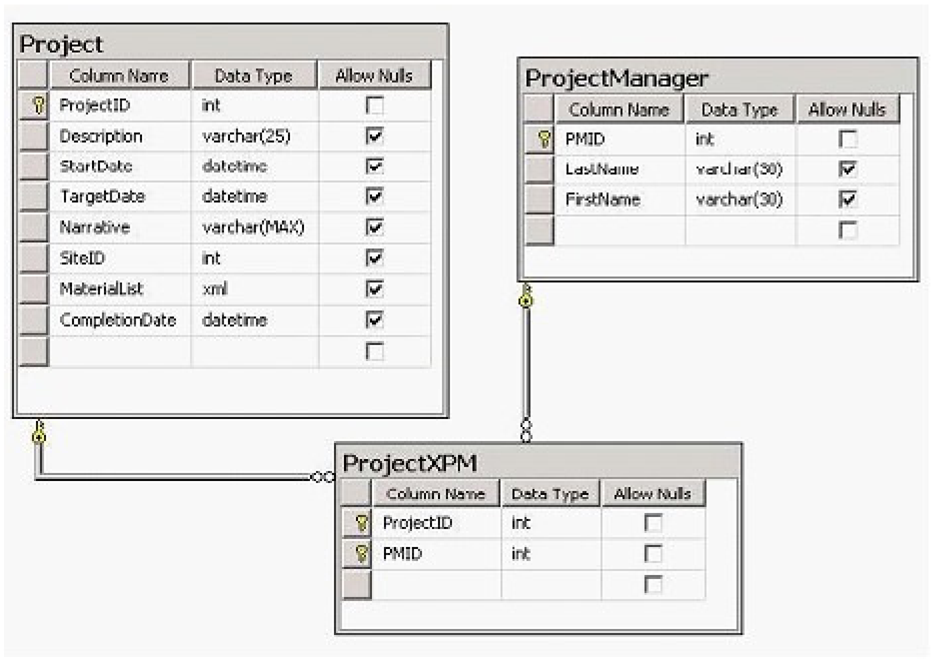
A. Shared the incoming XML documents into comma-delimited format, use a Bulk Import to load the data into database tables, and then use an SSIS package to extract the data from the database, validate it, and generate XML documents for vendors
B. Create and populate an **xml** data type column to store the incoming documents. Use stored procedures to extract the vendor documents and validate them using the given XSD.
C. Store the incoming XML documents as large objects. Use stored procedures to extract, validate, and output XML documents for vendors.
D. Extract the data from the incoming data files and store each element in a different column, and then use DML triggers to extract and validate the output files.

Answer: B
Section: (none)

Explanation/Reference:
An **xml** data type stores XML documents or document fragments in a column in a SQL Server table. You can use the **xml** data type as an input or output parameter of a stored procedure or as a variable or function return type. You can even use special `xml` methods to extract portions of the XML, as well perform many other XML-related tasks. Using an **xml** data type would also allow you to create an XML index on the column to improve query performance.
You should not shred the incoming XML documents into comma-delimited format, use a Bulk Import to load the data into database tables, and then use an SSIS package to extract the data from the database, validate it, and generate XML documents for vendors. Rather than perform excessive transformations on both incoming XML documents and output XML, using an **xml** data type allows you to store the XML data in the

database and use it without the extra effort.
You should not store the incoming XML documents as large objects and use stored procedures to extract, validate, and output XML documents for vendors. Storing the XML documents as large objects containing character data eliminates the possibility of using **xml** data type methods to query and manipulate the XML.
You should not extract the data from the incoming data files and store each element in a different column, and then use DML triggers to extract and validate the output files. Although you can shred XML documents and store it in the database in other data types, using an **xml** data type in this scenario would be most appropriate because you need to extract portions of the XML to output later. DML triggers fire automatically when a DML statement executes, and could not be called on demand to extract data as needed.

QUESTION 244
You are a database developer on an instance of SQL Server 2008. Your **Prod** database contains tables to maintain project-related information for construction projects undertaken by your company. A partial data model is included as follows:



Project

Column Name	Data Type	Allow Nulls
ProjectID	int	☐
Description	varchar(25)	✓
StartDate	datetime	✓
TargetDate	datetime	✓
Narrative	varchar(MAX)	✓
SiteID	int	✓
MaterialList	xml	✓
CompletionDate	datetime	✓
		☐

ProjectManager

Column Name	Data Type	Allow Nulls
PMID	int	☐
LastName	varchar(30)	✓
FirstName	varchar(30)	✓
		☐

ProjectXPM

Column Name	Data Type	Allow Nulls
ProjectID	int	☐
PMID	int	☐
		☐

The **MaterialList** column in the **Project** table contains an XML material list for each project in the following format:

```
<Materials>
<Item>  <SKU ID="XK-33" ItemDesc="Corner Brackets" /> <Quantity>15</Quantity>
<RequiredDt>03-15-2009</RequiredDt>
</Item>
<Item>  <SKU ID="XV-17" ItemDesc="Metal Shelving" /> <Quantity>100</Quantity>
<RequiredDt>03-12-2009</RequiredDt>
</Item></Materials>
```

You execute the following query:

```
SELECT MaterialList.query('<MasterList> {/Materials/Item/SKU}</MasterList>')FROM
Project;
```

Which output format would be displayed in the result set?

A. `<SKU ID="XV-17" ItemDesc="Metal Shelving" />`
`<SKU ID="XK-33" ItemDesc="Corner Brackets" />`

B. `<MasterList> <SKU ID="XK-33" ItemDesc="Corner Brackets" /><Quantity>15</Quantity><RequiredDt>03-15-2009</RequiredDt><SKU ID="XV-17" ItemDesc="Metal Shelving" /><Quantity>100</Quantity><RequiredDt>03-12-2009</RequiredDt></MasterList>`

C. `<MasterList> <Item> <SKU ID="XK-33" ItemDesc="Corner Brackets" /><Quantity>15</Quantity><RequiredDt>03-15-2009</RequiredDt></Item><Item> <SKU ID="XV-17" ItemDesc="Metal Shelving" /> <Quantity>100</Quantity><RequiredDt>03-12-2009</RequiredDt></Item></MasterList>`

D. `<MasterList> <SKU ID="XK-33" ItemDesc="Corner Brackets" /><SKU ID="XV-17" ItemDesc="Metal Shelving" /></MasterList>`

Answer: D
Section: (none)

Explanation/Reference:
The **query()** method is used to query and retrieve XML elements and attributes from an XML instance. The method accepts a string XQuery expression that determines which elements and element attributes are extracted. In this scenario, you executed the following query:

```
SELECT MaterialList.query('<MasterList> {/Materials/Item/SKU}</MasterList>')FROM
Project;
```

The string you specified included a literal `<MasterList>` element and its corresponding closing element. Within curly braces (`{ }`), you included an expression that specifies the absolute path used to retrieve the nodes. This code extracts the `<SKU>` element, which is a child of the `<Item>` element from within the `<Materials>` element.
The output format that includes only `<SKU>` elements is incorrect. In this scenario, you used literal values to enclose the output XML within a `<MasterList>` element. The output format that includes only `<SKU>` elements could be generated by the following query:

```
SELECT MaterialList.query('/Materials/Item/SKU')FROM Project;
```

The output format that contains no `<Item>` elements and multiple `<SKU>`, `<Quantity>`, and `<RequiredDt>` elements within each `<MasterList>` element is incorrect. This output could be generated by the following query:

```
SELECT MaterialList.query('<MasterList> {/Materials/Item/*}</MasterList>')FROM
Project;
```

The output format that contains `<Item>` elements within the `<MasterList>` element and has subelements within each `<Item>` element is incorrect. This output could be generated by the following query:

```
SELECT MaterialList.query('<MasterList> {/Materials/*}</MasterList>')FROM
Project;
```

QUESTION 245
You are a database developer on an instance of SQL Server 2008. You need to add a column to your **AssemblyDetail** table to hold a date and time value. You want the time portion to include to highest data precision possible, but you do not want to store time zone information.
Which data type should you use?

A. smalldatetime
B. datetimeoffset
C. datetime2
D. datetime

Answer: C
Section: (none)

Explanation/Reference:
You should use a **datetime2** data type. The **datetime2** data type is a new SQL Server 2008 data type that is similar to a **datetime** data type, but has greater range and precision. The **datetime2** data type stores a date and a time that includes fractional seconds up to seven digits of precision (YYYY-MM-DD hh:mm:ss.0000000).
You should not use a **smalldatetime** data type. The **smalldatetime** data type stores both the date and the

time, without including any fractional seconds.
You should not use a **datetimeoffset** data type because in this scenario you did not want to store time zone information. The **datetimeoffset** data
type includes time zone information as a time zone offset. The time zone offset, which is in the format of +hh: mm or -hh:mm, ranges from -14:00 to +14:00. The time zone offset indicates the difference between Coordinated Universal Time (UTC) and the local time. For a positive time zone offset, the local time is calculated by adding the time zone offset to UTC. For a negative time zone offset, the local time is calculated by subtracting the time zone offset from UTC. For example, a time zone offset of -11:00 indicates that the date/time uses a local time zone that is eleven hours behind UTC.
You should not use a **datetime** data type because a **datetime2** data type has greater range and precision. A **datetime** data type has only three digits to represent fractional seconds of the time.

QUESTION 246
You are the database administrator for a manufacturing company. Your company sells the products it manufactures to wholesale distributors and directly to consumers. Your company wants to create a data warehouse to analyze past sales revenue, analyze key performance indicators, predict products that will be popular with new customers, and make decisions about discontinuing specific product lines.
SQL Server 2008 Enterprise Edition is installed on the server computer named **Srvr1**. At weekly intervals, you must load the data from the following data stores into staging tables of the **DW1** database on the server:
Customer demographic data and order data that is captured and maintained in an Oracle database using an Oracle-based order entry system
Historical customer sales data stored in SQL Server tables
Historical records of machine downtime that are captured manually by machine operators and stored in Microsoft Excel

Specific rules must be enforced when loading data into the staging tables so that only valid data that does not already exist in the data warehouse is transferred.
Which component should you use to load the data?

A. SQL Server Reporting Services (SSRS)
B. SQL Server Analysis Services (SSAS)
C. SQL Server Integration Services (SSIS)
D. SQL Server Notification Services (SSNS)
E. Microsoft Sync Framework

Answer: C
Section: (none)

Explanation/Reference:
SSIS can create packages to retrieve data from multiple data sources, such as the Oracle database, SQL Server tables, and Microsoft Excel in this scenario. You can then transform the data as needed into a standard format, cleanse it, and load it into a variety of data sources, such as the staging tables in this scenario. During the transfer process, you can transform, validate, cleanse, and manipulate the data as needed, or combine data from multiple data stores. For example, you can aggregate or sort data, convert data types, or create new calculated columns from the existing data. In addition to SSIS' traditional ETL function, SSIS also can be used to access data on demand from different data sources. For example, SSIS provides a DataReader destination you can use as a data source to access data using a DataReader. With SSIS, you can often bypass or minimize staging. SSIS also exposes .NET Framework and native APIs that you can use to extend SSIS's functionality or integrate with legacy systems or third-party products.
You should not use SQL Server Analysis Services (SSAS) to load the data. SSAS provides tools to create and maintain SSAS databases. You can use SSAS to define cubes and dimensions and to perform complex data analysis, including defining and analyzing key performance indicators and making business predictions based on the data. In this scenario, after the data is loaded into the staging areas with SSIS, you could use SSAS to create the data warehouse, analyze past sales revenue and key performance indicators, predict products that will be popular with new customers, and make decisions about discontinuing specific product lines.

You should not use SQL Server Reporting Services (SSRS) to load the data. SSRS is a SQL Server component used to create, generate, deliver, and process reports from a variety of data sources, such as relational data, multidimensional data, or XML. You can create traditional reports directly from a data source or from SSAS databases. You can create reports in a variety of formats, export reports to other applications, allow users to interactively drill down through reports to explore the data, provide scheduled or on-demand reports, and even programmatically integrate reporting with custom applications or meet complex reporting requirements. SSRS also provides security features to allow only authorized users to access reports.
SSIS, SSAS, and SSRS can be used together to create complex Business Intelligence (BI) applications using SQL Server Business Intelligence Development Studio (BIDS).
You should not use SQL Server Notification Services (SSNS). SSNS provides support for notification applications in previous SQL Server versions. SSNS allows you to generate and send notifications to database users. With SQL Server 2008, notification support is incorporated into SSRS.
You should not use Microsoft Sync Framework. The Microsoft Sync Framework supports collaboration by providing services and tools to synchronize databases, files, and files systems, provide roaming support, and provide access to offline data.

QUESTION 247
You are a database developer. You plan to design a database solution by using SQL Server 2008. You create a stored procedure that uses the TRY/CATCH syntax in a new database. When the stored procedure is executed, it logs information about each step in the TRY block into a table named dbo.ExecutionLog. When an error occurs, the stored procedure must perform the following tasks:

- Roll back the changes made to the target tables.
- Retain the log entries stored in the dbo.ExecutionLog table.

You need to ensure that the stored procedure performs the given tasks. What should you do?

A. 1.Start a transaction in the TRY block.
   2. After each step, insert log entries into the dbo.ExecutionLog table.
   3. In the CATCH block, commit the transaction.
   4. After the CATCH block, use data in the dbo.ExecutionLog table to reverse any changes made to thetarget tables.
   5. Commit the transaction if one exists.

B. 1. Start a transaction in the TRY block.
   2. Before each step, define a transactional save point.
   3. After each step, insert log entries into the dbo.ExecutionLog table.
   4. In the CATCH block, roll back to the transactional save points.
   5. After the CATCH block, commit the transaction.

C. 1. Define a temporary table before the TRY block by using the same columns as that of the dbo. ExecutionLog table.
   2. Start a transaction in the TRY block.
   3. After each step, insert log entries into the temporary table.
   4. In the CATCH block, roll back the transaction.
   5. After the CATCH block, insert the rows from the temporary table into the dbo.ExecutionLog table.
   6. Commit the transaction if one exists.

D. 1. Define a table variable before the TRY block by using the same columns as that of the dbo. ExecutionLog table.
   2. Start a transaction in the TRY block.
   3. After each step, insert log entries into the table variable.
   4. In the CATCH block, roll back the transaction.
   5. After the CATCH block, insert the rows from the table variable into the dbo.ExecutionLog table.
   6. Commit the transaction if one exists.

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 248
You are a database developer on an instance of SQL Server 2008. You receive input from several business partners as XML documents. You process the incoming XML documents and store the data in the**InData** table in an **xml** data type column.
You have a stored procedure named **ProcessUpdate** that includes a transaction that performs the following tasks:
Updates several rows in a lookup table
Updates rows in the **Master** table based on the **xml** column value in the **InData** table
Deletes rows from the **InData** table for rows successfully processed You expect a high volume of incoming XML files and are concerned about performance of the **ProcessUpdate** stored procedure. You want to ensure that queries do not time out or cause lock escalation if possible. Which action should you take?

A. Store the incoming XML documents on the file system instead of in the database.
B. Modify the **ProcessUpdate** stored procedure to execute multiple transactions, with each transaction processing a smaller batch of incoming
XML documents.
C. Set the transaction isolation level to `SERIALIZABLE`.
D. Include the `TABLOCK` table hint on the query that updates the **Master** table.

Answer: B
Section: (none)

Explanation/Reference:
You should modify the **ProcessUpdate** stored procedure to execute multiple transactions, with each transaction processing a smaller batch of incoming XML documents. When a transaction is processing a large volume of rows and performing a large number of DML operations, it often adversely affects performance. Transactions should be as short as possible to minimize locking issues for other users. When the volume is high, you can split the work of the transaction into smaller batches and commit each of the smaller batches as they are processed. This will reduce the memory required for processing and minimize blocking and contention issues for other database users.
You should not store the incoming XML documents on the file system instead of in the database. Using FILESTREAM storage would reduce the database size, but the transaction would still process all of the data, and would likely cause concurrency issues when performing the updates and deletes.
You should not set the transaction isolation level to `SERIALIZABLE` because this would likely cause concurrency issues. The `SERIALIZABLE` transaction isolation level is the most restrictive and completely isolates transactions from one another. Because you may be processing a large number of rows, this could adversely affect other users accessing the **Master** table.
You should not include the `TABLOCK` table hint on the query that updates the **Master** table. The `TABLOCK` table hint can be specified within a query to lock a specific table while the query executes. In this scenario, you have a transaction that is performing multiple DML tasks, and locking the table during one of the transaction's activities would not improve overall performance for the transaction.

QUESTION 249
You need to design a method for storing large XML-formatted data. The design must meet the following requirements:
·Minimize the page I/O
·Minimize the response time for data manipulation language (DML) queries What should you do?

A. Store the XML data by using the filestream data type.
B. Store the XML data by using the nvarchar(max) data type.
C. Create columns based on XML elements. Shred the XML data into the individual columns.

D. Create columns based on Extensible Stylesheet Language Transformations (XSLT). Store the XML data by using the XML data type.

Answer: C
Section: (none)

Explanation/Reference:
Certkey.com - Make You Succeed To Pass IT Exams
Certkey 70-451

QUESTION 250
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database. A database includes a table named Selling. Client order summary information is contained in the Selling table. A stored procedure in created and it uses a SELECT statement and it must return a precise summation of the total sales for the current day when executing. When the stored procedure is being executed, you have to prevent any data modification in the Selling table by using a query hint. So which query hint should you use?

A. TABLOCKX
B. READPAST
C. HOLDLOCK
D. READCOMMITTED

Answer: A
Section: (none)

Explanation/Reference:

QUESTION 251
You are a database developer on an instance of SQL Server 2008. Your database contains a **DisciplinaryAction** table created with the following statement:
```
CREATE TABLE DisciplinaryAction (ActionID int,EmpID int, 7
ReportingBy int,ActionDate datetime,ActionDetails varchar(4000),CONSTRAINT
PK_ActionID PRIMARY KEY CLUSTERED (ActionID));
```
You have created a full-text catalog and full-text index to enable full-text searches on the **ActionDetails** column.
You want to identify all employees who have a disciplinary action relating to attendance issues. You also want the results to be ranked according to their relevance.
Which construct should you use?

A. the `CONTAINS` predicate including the `NEAR` clause
B. the `CONTAINSTABLE` function including the `FORMS OF INFLECTIONAL` clause
C. the `FREETEXT` predicate
D. the `FREETEXTTABLE` function

Answer: D
Section: (none)

Explanation/Reference:
The `FREETEXTTABLE` function can be used in the `FROM` clause of a query to search for word or phrase's meaning instead of the exact word or phrase. The `FREETEXTTABLE` function returns a table containing rows

matching the specified selection criteria. Each row of the table contains a **Key** column and a **Rank** column. The **Key** column contains the full-text key, and the **Rank** column contains a relevance ranking. Thesaurus files are used with queries that use the FREETEXT predicate and the FREETEXTTABLE function, and with queries that use the CONTAINS predicate and the CONTAINSTABLE function with the FORMS OF THESAURUS clause. You can customize thesaurus files to include desired synonyms.
You should not use the CONTAINS predicate including a NEAR clause. The CONTAINS predicate can be used to search character-based columns for an inflectional form of a specific word, a specific word or phrase, a word or phrase that is near another word or phrase, or words or phrases with weighted values. Using theNEAR clause searches for words or phrases that are near other words or phrases. In addition, theCONTAINS predicate does not return a relevance ranking as required in this scenario.
You should not use the CONTAINSTABLE function including the FORMS OF INFLECTIONAL clause. Using the FORMS OF INFLECTIONAL clause with the CONTAINS predicate or the CONTAINSTABLE function searches for a word and any other forms of that word, but does not search for meaning based on synonyms in the thesaurus.
You should not use the FREETEXT predicate. Although you can use the FREETEXT predicate to search for matches based on meaning rather than the exact words, the FREETEXT predicate does not return a relevance ranking as required in this scenario. The FREETEXT predicate is used in a WHERE clause to search full-text enabled character-based columns for values that match the meaning and not the exact wording. The FREETEXT predicate accepts two parameters. The first parameter specifies one or more full-text enabled columns to search. An asterisk (*) in the first parameter indicates that all full-text enabled columns should be searched. The second parameter is a character string that can contain words or phrases. If the search should be performed on phrases, the search string should be enclosed in double quotation marks instead of single quotation marks. With the FREETEXT predicate, strings are first separated into words, word-stemming is performed to determine other word forms, and then the result is passed through the thesaurus to identify words with similar meanings. The primary difference between using the thesaurus with the CONTAINS predicate and using the FREETEXT predicate is that CONTAINS only uses the thesaurus and does not perform word stemming.

QUESTION 252
You are a database developer on an instance of SQL Server 2008. You are creating a user-defined function to be used by the human resources department to identify the employee with the highest salary who meets other specific criteria.
You use the following statement to create the UDF:
```
CREATE FUNCTION dbo.udf_find_emp(@Commission money, @Status varchar(8) ='A')
 RETURNS int
WITH SCHEMABINDING, ENCRYPTION
AS
BEGIN
 DECLARE @v_emp int;
 SELECT @v_emp = EmpID
  FROM dbo.Employee
  WHERE Status = @Status
   AND Commission > @Commission
   AND Salary =(SELECT MAX(Salary)
                   FROM dbo.Employee
                   WHERE Status = @Status AND Commission > @Commission)
 ;
 RETURN @v_emp
END
;
```
Which statement about the function is true?

A. The function can only be called from within the body of a stored procedure.
B. The function will never return a NULL value.
C. The function encrypts all parameter values passed to the function.

D. To call the function successfully without specifying the `@Status` parameter value, you must include the `DEFAULT` keyword in the function call.

Answer: D
Section: (none)

Explanation/Reference:
In this scenario, you create a user-defined scalar function using the `CREATE FUNCTION` statement. The basic syntax of the `CREATE FUNCTION` statement for scalar functions is as follows:

```
CREATE FUNCTION [schema_name.]function_name ([{@parm_name [AS][parmtype_schema.]
parm_data_type [=default] [READONLY]} [,...n]])RETURNS return_type [WITH
function_opt [,...n]][AS]
BEGIN
function_body RETURN scalar_expression END;
```

When creating the function, you defined a default value for the `@Status` parameter. Therefore, to successfully call the function without an `@Status`value specified, you must also specify the `DEFAULT` keyword. When calling stored procedures, you can omit the `DEFAULT` keyword, but for user-defined functions, you cannot. For example, in this scenario, the following statement would execute successfully:

```
SELECT dbo.udf_find_emp(1000, DEFAULT);
```

The options that state that the function can only be called from within the body of a stored procedure, and that the function encrypts all parameter values passed to the function, are both incorrect. There are no options to specify when you are creating a function to restrict where a function maybe called from or to encrypt parameter values. When you create a scalar user-defined function, the function can be used anywhere that a scalar value could be used. In this scenario, you specified `WITH SCHEMABINDING, ENCRYPTION`. The `SCHEMABINDING` option is used to ensure that no objects on which the function depends are modified in a way that might make the function unusable. The `ENCRYPTION` option will encrypt the function's definition to ensure that it is not stored in plain text that is readable by others.
The option that states the function will never return a `NULL` value is incorrect. Even though the return type is defined as an `int`, the function may still return a `NULL` value if the `SELECT` statement returns no rows.

QUESTION 253
You are a database developer on an instance of SQL Server 2008. The **Name** and **Description** columns of the **AssemblyProcess** table are full-text enabled. You issue the following `SELECT` statement:

```
SELECT ID, DescriptionFROM AssemblyProcessWHERE CONTAINS(*, ' "weld" AND "grind"
');
```

What is returned in the result?

A. the **ID** and **Description** values for all rows containing the words weld and grind near each other in the full-text enabled columns
B. the **ID** and **Description** values for all rows with a **Name** containing the word weld or the word grind
C. the **ID** and **Description** values for all rows containing the words weld and grind in the full-text enabled columns
D. the **ID** and **Description** values for all rows containing the phrase "weld and grind" in the full-text enabled columns

Answer: C
Section: (none)

Explanation/Reference:
The `CONTAINS` predicate can be used to search character-based columns for an inflectional form of a specific word, a specific word or phrase, a word or phrase that is near another word or phrase, or words or phrases with weighted values. You can combine several conditions using the `AND`, `OR`, and `NOT` keywords. The `CONTAINS` search condition is case insensitive. The syntax for the `CONTAINS` predicate is as follows:

```
CONTAINS ({column | *}, '<contains_search_condition>')
```

The search condition is enclosed in single quotes, but individual words or phrases inside the search condition can be offset with double quotes. If the condition only includes single quotes, the argument is passed as a single search condition. A search condition of `' "weld" AND "grind" '`returns all rows having both weld and grind in the column. In contrast, a search condition of `'weld AND grind'` returns all rows having weld and grind separated by a stopword. Because stopwords, or noise words, are not recognized, the search condition of `'weld OR grind'` would produce an identical result.
The given statement does not return the **ID** and **Description** values for all rows containing the words weld and grind near each other in the full-text enabled columns. To perform such a search, you would use the `NEAR` keyword in the `CONTAINS` predicate.
The given statement does not return the **ID** and **Description** values for all rows with a **Name** containing either the word weld or grind. Using an asterisk (`*`) in the first argument of the `CONTAINS` predicate specifies that all columns registered for full-text searching be used.
The given statement does not return the **ID** and **Description** values for all rows containing the phrase "weld and grind" in the full-text enabled columns.

QUESTION 254
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. There's a query which is used very often. But you notice that this query executes quite slow.
This query often uses full-table scans while not indexes.
Because of this, other queries that modify the table are to be blocked.
On the underlying tables,the indexing strategy that the query uses can change.

Now you have to design a solution which removes full-table scans and allows the query optimizer to select the appropriate index.

So what should you do?

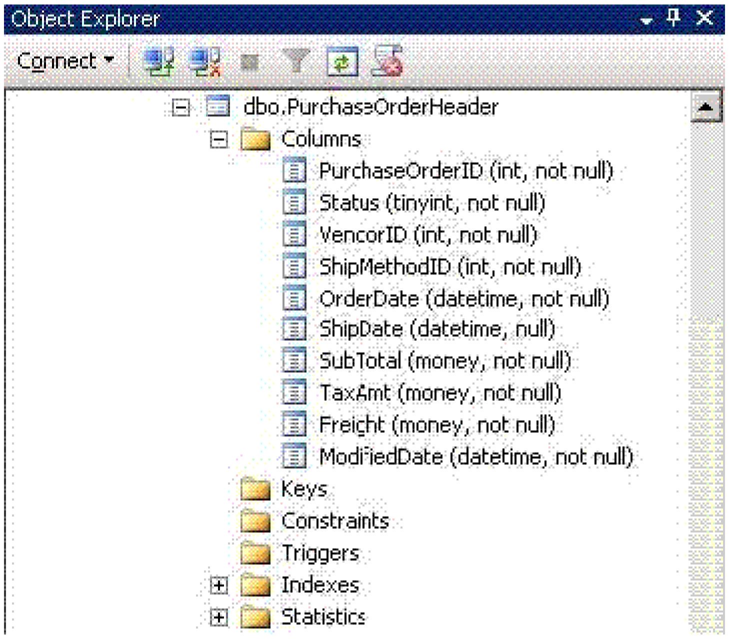
A. The INDEX table hint has to be used.
B. The NOEXPAND table hint has to be used.
C. The INDEX(0) table hint has to be used.
D. The FORCESEEK table hint has to be used.

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 255
Your database contains a **PurchaseOrderHeader** table as shown in the Object Explorer pane in the exhibit. (Click the **Exhibit(s)** button.)
In one session, a user executes the following Transact-SQL:
```
BEGIN TRANSACTION
UPDATE PurchaseOrderHeader SET ShipMethodID = 2
WHERE PurchaseOrderID = 1;
```
You issue the following query in another session, but the query hangs:
```
SELECT ShipMethodID, SUM(Freight)FROM PurchaseOrderHeader GROUP BY ShipMethodID;
```
What should you add to your query to allow it to execute immediately without errors?

Exhibit:



dbo.PurchaseOrderHeader
- Columns
  - PurchaseOrderID (int, not null)
  - Status (tinyint, not null)
  - VencorID (int, not null)
  - ShipMethodID (int, not null)
  - OrderDate (datetime, not null)
  - ShipDate (datetime, null)
  - SubTotal (money, not null)
  - TaxAmt (money, not null)
  - Freight (money, not null)
  - ModifiedDate (datetime, not null)
- Keys
- Constraints
- Triggers
- Indexes
- Statistics

A. the `NOWAIT` table hint
B. the `TABLOCK` table hint
C. the `READPAST` table hint
D. the `UPDLOCK` table hint

Answer: C
Section: (none)

Explanation/Reference:
In this scenario, another user has started a transaction that updates the **PurchaseOrderHeader** table and has issued an `UPDATE` statement, but has not committed the changes. The `UPDATE` statement is currently locking one row in the **PurchaseOrderHeader** table. This will cause your query to wait until the user has committed or rolled back the changes. To avoid this, you should add the `READPAST` table hint. When you include the `READPAST` table hint in a query, the query executes immediately, but does not return any locked rows. In this scenario, you could use the following statement to execute the query immediately without any errors:
```
SELECT ShipMethodID, SUM(Freight)FROM PurchaseOrderHeader WITH(READPAST)GROUP BY
ShipMethodID;
```
You should note that if you include the `READPAST` table hint, the **Freight** value of the row that is locked would not be included in the sum returned in your query's result set. In this scenario, if you wanted to execute the query and include the **Freight** value for the locked row, you could use the `NOLOCK` table hint. The `NOLOCK` table hint ignores all locking, but retrieves uncommitted data for locked rows. The `NOLOCK` table hint prevents blocking but allows dirty reads, and the `READPAST` table hint prevents blocking but does allow dirty reads.
You should not add the `NOWAIT` table hint because this would cause your query to immediately time out with the following error:
```
Msg 1222, Level 16, State 45, Line 1Lock request time out period exceeded.
```
The `NOWAIT` table hint will cause the query to not wait for locks to be released, and will immediately return an error. Usually, queries use the `LOCK_TIMEOUT` setting value to determine how long the statement should wait for locks to be released before returning an error. Using the `NOWAIT` table hint would be the equivalent of setting `LOCK_TIMEOUT` to 0 before executing the query.
You should not add the `TABLOCK` or `UPDLOCK` hint to your query because both of these hints would cause the query to wait to obtain locks. The `TABLOCK` hint specifies that the query should acquire a table-level lock and keep the table locked until the statement completes. The `UPDLOCK` hint specifies that the query should acquire an update lock that remains in effect until the transaction ends.

QUESTION 256
You have the following Transact-SQL code:
```
DECLARE @t int;SET @t = 9
SELECT * From SalesHeader WHERE TerritoryID = @tOPTION (OPTIMIZE FOR (@t=7);
```
What is the result?

A. The optimizer optimizes the query using parameter sniffing.
B. The optimizer uses a parameter value of 7 when optimizing the query.
C. The optimizer uses a parameter value of 9 when optimizing the query.
D. An error occurs because different values were specified for `@t`

Answer: B
Section: (none)

Explanation/Reference:
The `OPTIMIZE FOR` query hint forces the optimizer to use a specific parameter value when optimizing the query. In this scenario, you specified `OPTIMIZE FOR (@t=7)`. Therefore, the optimizer will use a parameter value of 7 when optimizing the query.
The optimizer does not optimize the query using parameter sniffing. Parameter sniffing occurs when SQL Server detects the current parameter value during compilation or recompilation, and passes it to the query optimizer. The query optimizer can then use this value to generate better execution plans. Parameter sniffing occurs automatically. However, in this scenario, you specified the `OPTIMIZE FOR` hint, which causes the optimizer to use a specific parameter value when optimizing the query.
The optimizer does not use a parameter value of 9 when optimizing the query. In this scenario, you set the value of `@t` to 9, but you override the parameter value the optimizer will use by specifying the `OPTIMIZE FOR` hint.
An error does not occur because different values were specified for `@t`. Even though the current value of `@t` is 9, you can specify a different value for `@t` in the `OPTIMIZE FOR` hint.

QUESTION 257
You are a database developer on an instance of SQL Server 2008. You execute the following Transact-SQL:
```
DECLARE @iHndl intDECLARE @XmlDoc nvarchar(4000)SET @XmlDoc = N'<Exams><Exam
ID="70-451" Desc="PRO: Designing Database Solutions and Data Access Using
Microsoft SQL Server 2008"><Obj ObjDesc="Designing a Database Strategy" /><Obj
ObjDesc="Designing Database Tables" /><Obj ObjDesc="Designing Programming
Objects" /><Obj ObjDesc="Designing a Transaction and Concurrency Strategy" /><Obj
ObjDesc="Designing an XML Strategy" /><Obj ObjDesc="Designing Queries for
Performance" /><Obj ObjDesc="Designing a Database for Optimal Performance" /></
Exam><Exam ID="70-433" Desc="TS: Microsoft SQL Server 2008, Database
Development"><Obj ObjDesc="Implementing Tables and Views" /><Obj
ObjDesc="Implementing Programming Objects" /><Obj ObjDesc="Working with Query
Fundamentals" /><Obj ObjDesc="Applying Additional Query Techniques" /><Obj
ObjDesc="Working with Additional SQL Server Components" /><Obj ObjDesc="Working
with XML Data" /><Obj ObjDesc="Gathering Performance Information" /></Exam></
Exams>'
EXEC sp_xml_preparedocument @iHndl OUTPUT, @XmlDoc;SELECT * FROM OPENXML (@iHndl,
'/Exams/Exam/Obj', 1)WITH (ExamID varchar(10) '../@ID',ObjDesc varchar(100));EXEC
sp_xml_removedocument @iHndl;
```
What is the result?

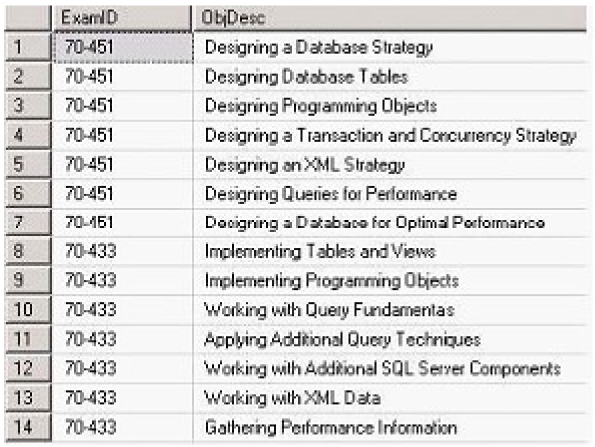
A. The query returns output in XML format containing the exam ID and the objective description.
B. The query returns output in a tabular result set containing the exam ID and the objective description.
C. The query returns output in tabular format containing the exam ID, the exam description, and the objective description.

D. The query generates an error because the `FROM` clause is incorrect.

Answer: B
Section: (none)

Explanation/Reference:
The query returns output in a tabular result set containing the exam ID and the objective description. In this scenario, you use the `OPENXML` function. `OPENXML` is a rowset provider function that creates a relational view of data contained in an XML document. `OPENXML` can be used in place of a table or view in a `SELECT` statement. The `OPENXML` function accepts the following parameters:
`idoc`: Identifies an integer document handle returned using the **sp_xml_preparedocument** system stored procedure.
`rowpattern`: Specifies the name of the node from which processing should begin.
`flag`: Identifies the type of the mapping as either attribute-specific or element-specific. A `flag` value of 1 indicates that the mapping should be attribute-specific, and a value of 2 indicates that the mapping should be element-specific.
`colpattern`: Specifies an XPath pattern to identify the XML elements that should be mapped. This parameter is used with the `WITH` clause of the query.

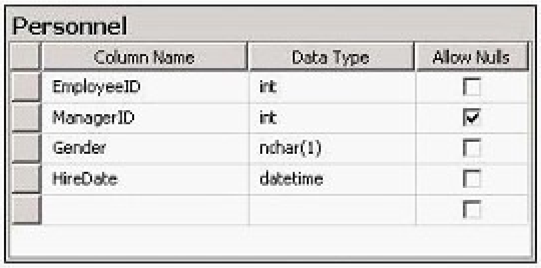
In this scenario, you first call the **sp_xml_preparedocument** system stored procedure to return the document handle. Then, you include a query that contains the `OPENXML` function in the `FROM` clause. In this scenario, `'/Exams/Exam/Obj '` specifies that processing should begin at the `<Obj>` node. Then, the `WITH` clause specifies the columns that should appear in the output. The columns will be output in the order they are specified. For each column in the `WITH` clause, you specify the column name and data type, and an optional XPath expression that specifies how mapping should occur. If you already have a table that has the desired columns, you can specify a table name instead of individually naming the columns. After the data has been extracted from the XML document, you call the **sp_xml_removedocument** stored procedure to remove the internal document handle and free resources. The Transact-SQL code in this scenario will return the following result set:



	ExamID	ObjDesc
1	70-451	Designing a Database Strategy
2	70-451	Designing Database Tables
3	70-451	Designing Programming Objects
4	70-451	Designing a Transaction and Concurrency Strategy
5	70-451	Designing an XML Strategy
6	70-451	Designing Queries for Performance
7	70-451	Designing a Database for Optimal Performance
8	70-433	Implementing Tables and Views
9	70-433	Implementing Programming Objects
10	70-433	Working with Query Fundamentas
11	70-433	Applying Additional Query Techniques
12	70-433	Working with Additional SQL Server Components
13	70-433	Working with XML Data
14	70-433	Gathering Performance Information

The option that states the query returns output in XML format is incorrect because the output is a tabular result set, not XML.
The option that states the query returns output in tabular format containing the exam ID, the exam description, and the objective descriptions is incorrect. This code only includes the value of the `<Exam>` element's `ID` attribute and the value of the `<Obj>` elements `ObjDesc` attribute in the result set.
The option that states the query generates an error because the `FROM` clause is incorrect. The `FROM` clause in this code correctly specifies the use of `OPENXML`.

QUESTION 258
You are a database developer on an instance of SQL Server 2008. Your **Personnel** table is defined as follows:



Personnel

	Column Name	Data Type	Allow Nulls
	EmployeeID	int	☐
	ManagerID	int	☑
	Gender	nchar(1)	☐
	HireDate	datetime	☐
			☐

The **Personnel** table has a primary key defined on the **EmployeeID** column and contains a table-level `CHECK` constraint that ensures that only the employee with an **EmployeeID** value of 1 may have a `NULL` value for the **ManagerID** column.
You execute the following Transact-SQL:

```
WITH MyCTE AS(SELECT EmployeeID, ManagerID, HireDateFROM Personnel WHERE
ManagerID IS NULLUNION ALL SELECT p.EmployeeID, p.ManagerID, p.HireDateFROM
Personnel pINNER JOIN MyCTE m ON m.EmployeeID = p.ManagerID)SELECT * FROM
MyCTEOPTION (MAXRECURSION 3);
```

What is the result?

A. The statement executes successfully with no errors and displays only three rows.
B. The statement fails because the CTE has not been defined as recursive.
C. The statement executes successfully with no errors and displays all employees, including each employee's **ManagerID** and **HireDate**.
D. The statement generates an error message, but displays employees that are at the first four levels of the organization chart.

Answer: D
Section: (none)

Explanation/Reference:
The query for a CTE definition can reference the original CTE. For example, in this scenario, the CTE definition references itself on the right side of the `INNER JOIN`. This causes the CTE to be recursive. The first `SELECT` statement in the CTE definition returns the employee with an **EmployeeID** value of 1, who is at the top of the organization chart because this employee does not have a manager. This query is referred to as an anchor member because it does not reference the CTE definition. The `SELECT` statement following the `UNION ALL` operator is referred to as a recursive member because it references the CTE definition. When creating a recursive CTE, forward referencing is not allowed. This means that the anchor member must be specified first.

In this scenario, you included the `UNION ALL` operator to combine the result of the first `SELECT` statement with the result of the second `SELECT` statement. You can also use the `UNION`, `INTERSECT`, or `EXCEPT` operator, but the `UNION ALL` operator must always be specified after the last anchor member before recursive members. Because these set operators are used, both of the `SELECT` statements must contain the same number
of columns, and the columns must have the same data type. In this scenario, the `SELECT` statement following the `UNION ALL` operator joins the original table with the result of the CTE. This joins each employee with the associated manager, continuing top-down through the organization chart with each recursive call.
In the statement that follows the CTE expression definition, you use the `MAXRECURSION` hint to limit the level of recursion to three levels. This means that the recursive query can only call itself three times. This traverses four levels of the organization chart: one for the top-level employee, and one for each recursion. In this scenario, when the recursion level maximum is exceeded, the result set up to that point is returned, but the statement terminates with a message similar to the following:

```
Msg 530, Level 16, State 1, Line 3The statement terminated. The maximum recursion
3 has been exhausted before statement completion.
```

You can use the `MAXRECURSION` hint to return only specific rows or prevent an infinite loop in a recursive CTE. If you do not specify a `MAXRECURSION` hint, recursion is limited to 100 levels by default. You can remove all maximum recursion restrictions by setting the `MAXRECURSION` hint to 0.

In addition, the recursive member of a CTE definition cannot contain certain constructs, such as the `DISTINCT` or `TOP` keywords, a `GROUP BY` or `HAVING` clause, or a reference to a subquery or aggregate function. Also, if a CTE is included in the definition of a view, the view is non-updateable.
The statement does not execute successfully with no errors and display only three rows. The statement might generate only three rows, but it could also display more than three rows, depending on the number of employees and organizational levels represented in the rows of the **Personnel** table. If the number of levels exceeded four, the statement would also generate an error message indicating the maximum recursion had been exceeded.
The statement does not execute successfully with no errors and display all employees, including each employee's **ManagerID** and **HireDate**. In this scenario, you specified a `MAXRECURSION` hint for the CTE. Selecting from the CTE may or may not display all employees, depending on the number of employees and organizational levels represented in the **Personnel** table. In addition, an error would be displayed if the maximum recursion were exceeded.

QUESTION 259
You are a database developer on an instance of SQL Server 2008. You have two partitioned tables with identical structures, named **Trx** and
Trx_Hy. They were both partitioned on the same column using the following partition function and partition scheme:

```
CREATE PARTITION FUNCTION Pf1 (int)AS RANGE LEFT FOR VALUES (10000, 30000,
50000);
CREATE PARTITION SCHEME Ps1 AS PARTITION Pf1 TO (fg1, fg2, fg3, fg4);
```

Both tables contain data. You issue the following statement to move a partition from the **Trx** table to the **Trx_Hy** table:

```
ALTER TABLE Trx SWITCH PARTITION 1 TO Trx_Hy PARTITION 4;
```

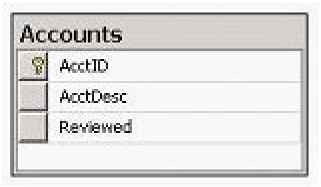
What is the result?

A. The statement fails if partition 4 of **Trx_Hy** contains data.
B. The statement succeeds if partition 1 of **Trx** and partition 4 of **Trx_Hy** both contain data.
C. The statement fails because the two tables use the same partition function.
D. The statement fails if either of the specified partitions is empty.

Answer: A
Section: (none)

Explanation/Reference:
The `ALTER TABLE` statement modifies a table definition to modify, add, or drop columns and constraints, switch partitions, or disable or enable triggers. The `ALTER TABLE...SWITCH` statement allows you to switch a partition from one partitioned table to another, which helps you transfer subsets of data quickly and efficiently. To be able to switch partitions, both tables must exist and be partitioned on the same column. In addition, the receiving partition must be empty. In this scenario, you issued a statement that would move partition 1 of the **Trx** table to partition 4 of the **Trx_Hy** table. This statement would fail if partition 4 of the **Trx_Hy** table contains data.
The option that states the statement fails if either of the specified partitions is empty is incorrect. To move a partition, the receiving partition must be empty.
The option that states the statement succeeds if partition 1 of **Trx** and partition 4 of **Trx_Hy** both contain data is incorrect. The statement will fail if partition 4 of the **Trx_Hy** table contains data.
The option that states the statement fails because the two tables use the same partition function is incorrect. Partitioned tables can use the same partition function. The partition function maps the rows of a table or index into partitions based on the specified partition boundary values.

QUESTION 260
You are a database administrator on an instance of SQL Server 2008. Your database contains the following **Accounts** table:



You execute the following Transact-SQL to create the **uspResetAccounts** stored procedure:

```
CREATE PROCEDURE dbo.uspResetAccounts AS
UPDATE dbo.Accounts SET Reviewed = NULL FROM dbo.Accounts;
ALTER TABLE dbo.Accounts ADD NextReview datetime;
```

You then grant **User1** permission to execute the **uspResetAccounts** stored procedure with the following statement:

```
GRANT EXECUTE ON uspResetAccounts TO User1;
```

User1 has no additional database permissions. **User1** issues the following statement:

```
EXECUTE dbo.uspResetAccounts;
```

What is the result?

A. The stored procedure executes successfully because ownership chaining occurs.

B. The stored procedure executes successfully only if cross-database ownership chaining is enabled.

C. The statement fails because **User1** does not have permission to execute the `ALTER TABLE` statement.

D. The statement fails because **User1** does not have permission to update the **Accounts** table.

Answer: C
Section: (none)

Explanation/Reference:
. In this scenario, you created a stored procedure and granted **User1** permission to execute the stored procedure. In this scenario, the stored procedure contains a DDL statement, `ALTER TABLE`. Therefore, an ownership chain is not created when **User1** executes the procedure. The statement will fail with the following error message:

```
Msg 1088, Level 16, State 13, Procedure uspResetAccounts, Line 10Cannot find the
object "Accounts" because it does not exist or you do not have permissions.
```

An ownership chain is created when a database object, such as stored procedure, accesses another database object, such as an underlying table. If a user has access to the stored procedure and the underlying table has the same owner as the stored procedure, then explicit permission to access the underlying table is not required. SQL Server will only check permissions on the underlying table if it has a different owner. However, you must note that ownership chaining does not occur when the stored procedure contains DDL. In this scenario, you could add the `EXECUTE AS` clause to the stored procedure definition to force the stored procedure to run with a specific execution context, regardless of who called it, and allow **User1** to execute the stored procedure successfully. You can also use the `EXECUTE AS` Transact-SQL statement to temporarily switch the execution context to another user's security context if necessary.
The option that states the stored procedure executes successfully because ownership chaining occurs is incorrect. Ownership chaining would occur if the **uspResetAccounts** stored procedure contained only DML statements. The stored procedure in this scenario contains a DDL statement, `ALTER TABLE`. If you modified the **uspResetAccounts** stored procedure so that it contained the `UPDATE` statement but not the `ALTER TABLE` statement, ownership chaining would occur, and **User1** would be able to execute the stored procedure without any errors.
The option that states the stored procedure executes successfully only if cross-database ownership chaining is enabled is incorrect. Cross-database ownership chaining allows you to use ownership chains across multiple databases, which is not required in this scenario.
The option that states the statement fails because **User1** does not have permission to update the **Accounts** table is incorrect. An ownership chain would be created because the **Accounts** table has the same owner as **uspResetAccounts** stored procedure. The `UPDATE` statement would execute successfully, but the `ALTER TABLE` statement would fail because an ownership chain is not created with DDL statements.

QUESTION 261
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database in a distributed enterprise environment. There're multiple servers in the enterprise environment.
All the servers have the same database which contains the table that has a surrogate key.
You must make sure that the surrogate key is unique across all servers;
Because of INSERT operations, the index on the surrogate key is not fragmented.

So what should you do to achieve this?

A. The timestamp data type should be used.
B. In a default constraint, the IDENTITY property should be used.Uniqueidentifier data type should be used.
C. In the column definition, the IDENTITY property should be used.The bigint data type should be used.
D. In a default constraint, the NEWSEQUENTIALID() function should be used.The uniqueidentifier data type should be used.

Answer: D
Section: (none)

Explanation/Reference:

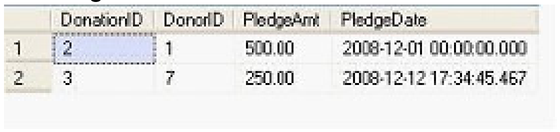
QUESTION 262
You are a database developer on an instance of SQL Server 2008. The **Donation** table in your database was created using the following statement:
```
CREATE TABLE Donation (DonationID int PRIMARY KEY,DonorID int,PledgeAmt money
CHECK (PledgeAmt > 0),PledgeDate datetime DEFAULT GETDATE());
```
The **Donation** table currently contains no data.
You execute the following Transact-SQL in SQL Server Management Studio:
```
SET XACT_ABORT ON BEGIN TRANSACTION INSERT INTO Donation VALUES(2, 1, 500, '12-
01-2008');INSERT INTO Donation VALUES(1, 3, 0, '12-15-2008'); INSERT INTO
Donation VALUES(3, 7, 250, DEFAULT);COMMIT TRANSACTION
```
What is the result?

A. The Transact-SQL executes successfully and inserts two rows into the **Donation** table.
B. The Transact-SQL executes successfully and inserts three rows into the **Donation** table.
C. The Transact-SQL generates an error message, but inserts two rows into the **Donation** table.
D. The Transact-SQL generates an error message and rolls back all inserts.

Answer: D
Section: (none)

Explanation/Reference:
When `XACT_ABORT` is set to `ON`, and a Transact-SQL statement raises an error at run time, the entire transaction is terminated and rolled back. When the `XACT_ABORT` option is set to `OFF`, only the Transact-SQL statement that raised the error is rolled back. The remaining statements in the transaction will be executed. The default value of the `XACT_ABORT` option is `OFF`. In this scenario, the second `INSERT` statement fails because it violates the `CHECK` constraint defined on the **PledgeAmt** column. The statement returns the following error message:
```
The INSERT statement conflicted with the CHECK constraint
"CK__Donation__Pledge__797309D9". The conflict occurred in database "KIT3", table
"dbo.Donation", column 'PledgeAmt'.
```
The entire transaction is rolled back, and no rows are inserted into the **Donation** table. If the `SET XACT_ABORT ON` statement had been omitted or `SET XACT_ABORT OFF` had been specified, the two

successful inserts would have been performed, and only the second insert would have been rolled back. The resulting table would be as follows:



	DonationID	DonorID	PledgeAmt	PledgeDate
1	2	1	500.00	2008-12-01 00:00:00.000
2	3	7	250.00	2008-12-12 17:34:45.467

All of the other options are incorrect because the entire transaction is rolled back.

QUESTION 263
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You use SQL Server 2008 to create a database solution in an enterprise environment.
You plan to create a stored procedure.
The procedure produces forecast data and queries a sales table.
You don't own the database and you do not have administrative permissions, but you are allowed to create stored procedures.
Users will only be allowed to execute your stored procedures.

What should you do to make sure that users can execute the stored procedures?

A. The TRUSTWORTHY property of the database should be set to ON.
B. When each stored procedure is created, you should include an EXECUTE AS OWNER clause.
C. When each stored procedure is created, you should include an EXECUTE AS CALLER clause.
D. In each stored procedure, before you query the sales table, you should include a SETUSER statement.

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 264
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. A table which contains information about Web pages is created by you.
And the Web pages are added to the Web sit which contains a home page and various other Web pages.
The home page is the root page of the site.
All pages except the root page have a link to an upper-level page.
the table you design must meet the following requirements:

Changing the links to the upper-level pages is a rare requirement;
Records of the Web pages that are linked to a particular page can be quickly retrieved.
The position of a Web page in a collection of linked pages can be quickly retrieved.

You have to make sure that the table is well designed.

So what should you use?

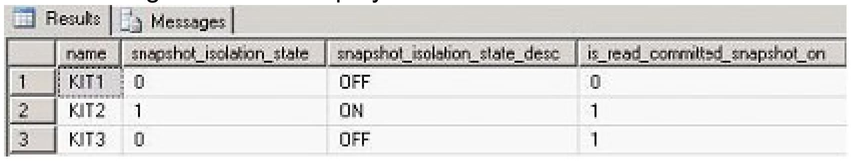
A. The XML data type should be used.
B. The hierarchyid data type should be used.
C. A Parent/Child mechanism that references the same table should be used.
D. A Parent/Child mechanism that references one or more additional tables should be used.

Answer: B

Section: (none)

Explanation/Reference:

QUESTION 265
Your SQL Server 2008 instance contains several databases. You execute the following Transact-SQL:
`SELECT name, snapshot_isolation_state, snapshot_isolation_state_desc, is_read_committed_snapshot_on FROM sys.databases;`
The following results are displayed:



Results | Messages

	name	snapshot_isolation_state	snapshot_isolation_state_desc	is_read_committed_snapshot_on
1	KIT1	0	OFF	0
2	KIT2	1	ON	1
3	KIT3	0	OFF	1

Which statement is true about your databases?

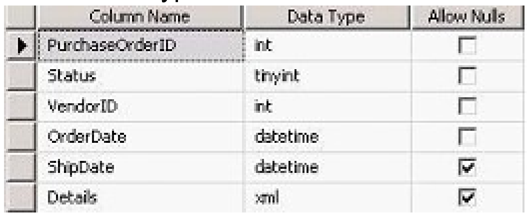
A. Transactions in the **KIT2** database can only specify a `READ COMMITTED` transaction isolation level.
B. Only transactions in the **KIT2** and **KIT3** databases may specify a `READ COMMITTED` transaction isolation level.
C. Transactions in the **KIT2** and **KIT3** databases with a `READ COMMITTED` isolation level will use row versioning instead of locking.
D. Transactions in **KIT2** with a transaction isolation level of `SNAPSHOT` will be able to access data in the **KIT3** database.

Answer: C
Section: (none)

Explanation/Reference:
The `READ_COMMITTED_SNAPSHOT` database option is used to control how transactions with a `READ COMMITTED` isolation level are handled. If the option is set to `ON`, which is the case based on the given output, row versioning is used instead of locking. When this setting is enabled, SQL Server stores the row versions in **tempdb** and uses the temporary row versions as they existed before starting the original transactions.
The option that states transactions in the **KIT2** database can only specify a `READ COMMITTED` transaction isolation level and the option that states only transactions in the **KIT2** and **KIT3** databases may specify a transaction isolation level of `READ COMMITTED` are incorrect. There is no database option that indicates that only `READ COMMITTED` transaction isolation levels are allowed. In this scenario, **KIT2** has both the `READ_COMMITTED_SNAPSHOT` and `ALLOW_SNAPSHOT_ISOLATION` database options set to `ON`. This indicates that for the **KIT2** database, row versioning is used for transactions with a `READ COMMITTED` transaction isolation level, and transactions with the `SNAPSHOT` isolation level are allowed. In this scenario, **KIT3** has `READ_COMMITTED_SNAPSHOT` set to `ON` and `ALLOW_SNAPSHOT_ISOLATION` set to `OFF`. This indicates that transactions on **KIT3** with a `READ COMMITTED` isolation level will use row versioning instead of locking, and transactions with the `SNAPSHOT` isolation level are not allowed.
The option that states only transactions in the **KIT2** and **KIT3** databases may specify a transaction isolation level of `READ COMMITTED` is incorrect. There is no database option that you can set to allow only transactions with a specified isolation level.
The option that states transactions in **KIT2** with a transaction isolation level of `SNAPSHOT` will be able to access data in the **KIT3** database is incorrect. To do so, both the **KIT2** and **KIT3** databases must have the `ALLOW_SNAPSHOT_ISOLATION` database option set to `ON`. In this scenario, it is only set to `ON` for the **KIT2** database.

QUESTION 266
You are a database developer on an instance of SQL Server 2008. You have a **POMaster** table that contains

an **xml** data type column named **Details** as shown:



	Column Name	Data Type	Allow Nulls
▶	PurchaseOrderID	int	☐
	Status	tinyint	☐
	VendorID	int	☐
	OrderDate	datetime	☐
	ShipDate	datetime	☑
	Details	xml	☑

An error occurred in the application that captures and records purchase details, and you need to update the **Details** column in the **POMaster** table. The **Details** column in the table currently contains the following XML for **PurchaseOrderID** 2006:

```
<POLines PONum="25" VendorName="VisionWorx"> <POLine> <InvID>026</
InvID><Quantity>7</Quantity><Cost>32.95</Cost><Taxable>1</Taxable></
POLine><POLine> <InvID>022</InvID>
<Quantity>142</Quantity><Cost>2.95</Cost><Taxable>1</Taxable></POLine></POLines>
```

You need to update the `VendorName` attribute in the XML to correct the error.

```
SET Details.modify('insert(/POLines[@VendorName="VisionWorx Corporation"]/
@VendorName) [1] ')WHERE PurchaseOrderID = 2006;
```

Which statement will successfully update the `VendorName` attribute of the `<POLines>` element in the **Details** column?

A.
```
UPDATE POMaster
SET Details.modify('/POLines[@VendorName="VisionWorx"]/@VendorName)
="VisionWorx Corporation"')WHERE PurchaseOrderID = 2006;
```
B.
```
UPDATE POMaster
SET Details.modify('replace value of(/POLines[@VendorName="VisionWorx"]/
@VendorName)
```
C.
```
UPDATE POMaster
SET Details.modify('replace value of(/POLines[@VendorName="VisionWorx"]/
@VendorName) [1]with "VisionWorx Corporation" ')WHERE PurchaseOrderID = 2006;
```
D.
```
UPDATE POMaster SET Details.modify('/POLines[@VendorName="VisionWorx"]/
@VendorName) [1]with "VisionWorx Corporation" ')WHERE PurchaseOrderID = 2006;
```

Answer: C
Section: (none)

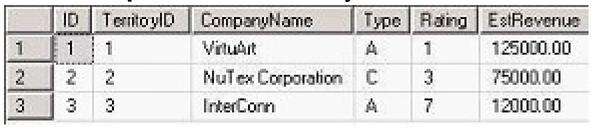
Explanation/Reference:
The **modify()** method is used to change the value of an **xml** type variable or column. The **modify()** method can only be invoked using the `SET` clause of an `UPDATE` statement. The method accepts a string argument containing an XML Data Manipulation Language (DML) statement. The statement is then used to update the corresponding element or attribute in the XML. In this scenario, the **modify()** method uses the **replace value of** XML DML statement to replace the `VendorName` attribute that contains the value "VisionWorx" with the value "VisionWorx Corporation". The **replace value of** XML DML statement has the following syntax:

```
replace value of
Expression1
with
Expression2
```

`Expression1` identifies the attribute or element to be updated, and `Expression2` identifies the new value. The `[1]` should be appended to the end of the expression to indicate that the target being updated is a single node. If you include an expression that identifies more than one node, an error occurs.

QUESTION 267
You are a database developer on an instance of SQL Server 2008. Your database contains a **Prospect** table defined using the following Transact-SQL:

```
CREATE TABLE Prospect(ID int IDENTITY(1,1) PRIMARY KEY,TerritoryID int NULL,
CompanyName varchar(35) UNIQUE,Type char(1) NOT NULL,Rating tinyint,EstRevenue
money);
```
Your **Prospect** table currently contains the following data:



	ID	TerritoyID	CompanyName	Type	Rating	EstRevenue
1	1	1	VirtuArt	A	1	125000.00
2	2	2	NuTex Corporation	C	3	75000.00
3	3	3	InterConn	A	7	12000.00

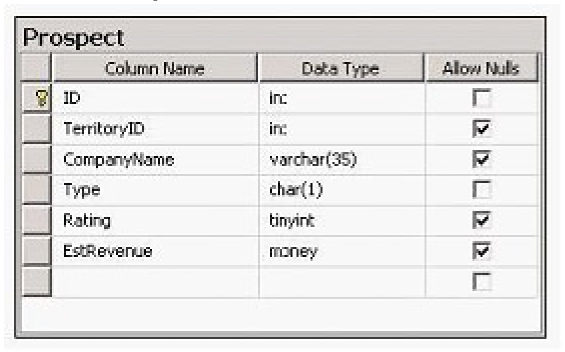
You use the following Transact-SQL statement to create a view:
```
CREATE VIEW ProspectViewAS SELECT * FROM ProspectWHERE Rating BETWEEN 1 and 3WITH
CHECK OPTION;
```
Assuming appropriate permissions, which two `UPDATE` statements will subsequently execute successfully and update a row in the **Prospect** table? (Choose two.)

A.
```
UPDATE Prospect
SET Rating = 9
WHERE ID = 1;
```
B.
```
UPDATE ProspectView
SET Rating = 5
WHERE ID = 3;
```
C.
```
UPDATE ProspectView
SET Rating = 2
WHERE ID = 2;
```
D.
```
UPDATE ProspectView
SET Rating = 1
WHERE ID = 3;
```
E.
```
UPDATE ProspectView
SET Rating = 8
WHERE ID = 2;
```

Answer: AC
Section: (none)

Explanation/Reference:
In this scenario, you created a view that included the `WITH CHECK OPTION` clause. The `WITH CHECK OPTION` clause is used to ensure that no
data modifications can be made through a view that would cause the underlying data to violate the view's definition. The first statement directly updates the **Prospect** table, and no `CHECK` constraint is defined on the **Rating** column of the **Prospect** table. Therefore, the statement executes and updates the **Rating** value for the first row in the table. The second statement updates the **Prospect** table using the **ProspectView** view. The row with an **ID** value of 2 is visible through the view, and the **Rating** value specified in the `SET` clause conforms to the `WHERE` clause of the view's definition. Therefore, the statement executes and updates the **Rating** value for the second row in the table.
The following two statements will execute successfully, but will not update a row in the **Prospect** table because the row with an **ID** value of 3 is not visible through the **ProspectView** view:
```
UPDATE ProspectViewSET Rating = 5WHERE ID = 3;
UPDATE ProspectViewSET Rating = 1WHERE ID = 3;
```
The following statement will generate an error because it uses the view to update the table, but attempts to set the **Rating** column to a value that does not conform to the `WHERE` clause of the view's definition:
```
UPDATE ProspectViewSET Rating = 8WHERE ID = 2;
```
If you executed this statement, the following message would be displayed:
```
Error message:Msg 550, Level 16, State 1, Line 1The attempted insert or update
failed because the target view either specifies WITH CHECK OPTION or spans aview
that specifies WITH CHECK OPTION and one or more rows resulting from the
operation did not
qualify under the CHECK OPTION constraint.
```

QUESTION 268
You are a database developer on an instance of SQL Server 2008. Your **Prod** database contains a table named **Prospect** that is defined as follows:



Prospect

Column Name	Data Type	Allow Nulls
ID	int	☐
TerritoryID	int	✓
CompanyName	varchar(35)	✓
Type	char(1)	☐
Rating	tinyint	✓
EstRevenue	money	✓
		☐

You are creating Transact-SQL code to remove obsolete prospects from the **Prospect** table. If the prospect is deleted from the **Prospect** table, the prospect's information should be displayed.
What should you do?

A. Use a cursor to iterate through the **Prospect** table and perform the required deletes, and then display the deleted rows.
B. Use a `DELETE` statement that includes an `OUTPUT` clause.
C. Query the **Prospect** table and perform the deletes within a `WHILE` loop. j
D. Query the **Prospect** table using a `SELECT...INTO` statement to populate a temporary table, and process the temporary table to perform the
deletions

Answer: B
Section: (none)

Explanation/Reference:
performing DML operations, you can use the `OUTPUT` clause to obtain and display information about affected rows. The `OUTPUT` clause can display this information to the user, insert the data into another permanent or temporary table or table variable using an `INTO` clause, or pass the data to a nested DML statement for processing.
Within the `OUTPUT` clause, you specify the column values to be retrieved by using the column names with the `INSERTED` and `DELETED` prefixes. Columns included in the `OUTPUT` clause must use one of the prefixes or an error is generated. For a `DELETE` statement, only the `DELETED` prefix is valid, and returns the column value that was deleted. An asterisk (`*`) indicates all columns in the table. In this scenario, you might use the following `DELETE` statement to delete prospects and display the deleted rows:
```
DELETE FROM ProspectOUTPUT DELETED.* WHERE Rating = 0;
```
All of the other options are incorrect. You can use the `OUTPUT` clause to accomplish the desired result using a set-based operation, rather than using a row-based approach or a temporary table. Using a set-based operation will provide better performance and minimize the complexity of the code.

QUESTION 269
You are a database developer. You plan to design a database solution by using SQL Server 2008.
The database contains a table named Claims. The structure of the Claims table is as shown in the following table.

Column Name	Description	Constraint
open_date	Claim received date	Does not allow the NULL value
close_date	Claim settled date	Allows the NULL value
status	• Set as **Open** when the claim is received • Set as **Closed** when the claim is settled	–

Only two percent of the claims are open at any point in time. You discover that queries on claims that have an Open status take a long time to execute. You need to optimize the performance of the claim-processing queries.
What should you do?

A. Use a partitioning function to partition the Claims table on the open_date column.
B. Create a view for the Claims table by using a WHERE clause to include all rows that have a NULL value in the close_date column.
C. Create an index for the Claims table by using a WHERE clause to include all rows that have a NULL value in the close_date column.
D. Create a table-valued function for the Claims table by using a WHERE clause to include all rows that have a NULL value in the close_date column.

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 270
You have a table named Table1. A sample of the data in Table1 is shown in the following table.

SalesID	SalesOrderNumber
1	SO2159
2	SO2768
3	SO3978
4	SO3010
5	SO4818
6	SO3919
7	SO3999

There is a defined nonclustered index on the SalesOrderNumber column.
The following query executes against the table.
WHERE `SO3' = LEFT(SalesOrderNumber,3)
You need to minimize the amount page I/O that is generated when the query runs.
What should you do?

A. Use a query hint.
B. Add a non-filtered index.
C. Rewrite the WHERE clause to use a LIKE statement.
D. Rewrite the WHERE clause to use a substring function on the SalesOrderNumber column.

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 271
You have a table that has an XML column named XMLOrderHeader. You need to design a stored procedure that extracts the order header values and stores them in a table. The solution must meet the following requirements:
·Extract many values
·Minimize the development effort
What should the solution include?

A. Use a single Exists() method.
B. Use a single XPATH statement.
C. For each value, use the Exists() method.
D. For each value, use an XPATH statement.

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 272
You are a database developer on an instance of SQL Server. You have the tables shown in the exhibit. (Click the **Exhibit(s)** button.)
These tables contain historical data, and each is populated at the end of each month by an SSIS package. For each table, a clustered index exists
on the primary key column. Additional nonclustered indexes exist on both tables.
You want to minimize storage space required by the **Invoices** and **InvoiceLines** tables. You want each column to use the fewest bytes necessary.
You want to accomplish this with minimal effort.
Which action should you take?

A. Use a sparse column set.
B. Implement FILESTREAM storage.
C. Compress the nonclustered indexes for both tables.
D. Implement row compression for both tables.

Answer: D
Section: (none)

Explanation/Reference:
You should implement row compression for both tables. Row compression saves storage space for each row by internally compressing each fixedlength
numeric, date/time, or character data type. For example, if you implemented row compression for the **Invoices** and **InvoiceLines** tables in
this scenario, columns in the tables would use only the space required, rather than using the actual number of bytes for each data type. For
example, if your company had only 200 sales representatives, with **SalesRepID**s between 1 and 200, each row would use only one byte to store the
SalesRepID instead of using the four bytes that an **int** data type requires. You can implement compression on a table when it is created or

afterwards using the `ALTER TABLE` statement. In this scenario, you could implement row compression for the **Invoices** and **InvoiceLines** tables
using the following statements:

```
ALTER TABLE Invoices
REBUILD WITH (DATA_COMPRESSION=ROW);
ALTER TABLE InvoiceLines
REBUILD WITH (DATA_COMPRESSION=ROW);
```

You can also implement compression at the page level by specifying the `DATA_COMPRESSION=PAGE` option. Page compression is implemented
internally by SQL Server to minimize the space required to store duplicated data. SQL Server uses a column-prefix or page-dictionary compression
technique to eliminate redundant data within each page. When you implement page compression, SQL Server implements also implements row
compression.
To meet the scenario requirements, you could also directly modify each of the tables' columns to use the data type that would minimize storage
based on the underlying data. However, this would take excessive effort, and might not be feasible if such table changes might break existing
applications. Using row compression provides the benefit of not having to modify the characteristics of each individual column, while still minimizing
the required storage.
Compression does have some associated overhead, especially if the table is frequently accessed or modified, because the data must be
compressed and uncompressed frequently. Therefore, you should closely evaluate not only the storage requirements but also how the table is used
before implementing compression. In this scenario, the data is rarely queried and modified. Therefore, the additional overhead would likely have no
adverse effects.
You should not use a sparse column set. Sparse columns are used to optimize the storage of `NULL` values when a table contains mostly `NULL`
values for a column. Column sets can be used with a table that contains sparse columns to be able to return all sparse column values as XML. In
this scenario, the details about the underlying data were not provided, and no columns were mentioned as containing mostly `NULL` values.
You should not implement FILESTREAM storage. FILESTREAM storage is implemented to store large binary objects, such as image or video files,
as files on the file system and be able manage to them using Transact-SQL. FILESTREAM data can also be accessed using Win32 APIs.
FILESTREAM storage is tightly integrated with most database functionality, including backup and recovery. When you take a database backup,
FILESTREAM storage is also backed up unless you override this functionality by performing a partial backup. To create a table that can store
FILESTREAM data, you create a table that contains a column of the **varbinary(max)** data type and include the `FILESTREAM` attribute.
You should not compress the nonclustered indexes for both tables. Although you can implement compression for indexes, doing so would not
minimize storage for the tables as required in this scenario. You might choose to compress indexes to minimize storage for an index. To compress
a nonclustered index named **IX_ProductID** on the **InvoiceLines** table, you could use the following statement:

```
ALTER INDEX IX_ProductID
ON InvoiceLines
REBUILD WITH (DATA_COMPRESSION=PAGE);
```

QUESTION 273
Users frequently update millions of rows in a table at a time. Users report that it takes a long time to update the rows. You need to recommend a solution to reduce the time it takes to update the rows. The solution must be developed in the minimum amount of time.

What should you do?

A. Use a table variable.
B. Use a temporary table.
C. Split the update operation into small batches.
D. Use the NOLOCK optimizer hint and use a single transaction.

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 274
You have a database that contains two tables. Both the XACT_ABORT database option and the IMPLICIT_TRANSACTIONS database option are set to OFF. You need to update both tables. If an update fails on either table, neither table should be updated. What should you do?

A. Use a transaction.
B. Change the isolation level.
C. Use the TABLOCK query hint.
D. Use the UPDLOCK query hint.

Answer: A
Section: (none)

Explanation/Reference:

QUESTION 275
You are a database developer on an instance of SQL Server 2008. You have a **WorkOrder** table and a **WorkOrderDetail** table. The **WorkOrder** table contains one row for each work order, and the **WorkOrderDetail** table contains line items of work performed for each work order.
You want to create a `FOREIGN KEY` constraint to relate the two tables. You want to ensure that if a work order identifier in the **WorkOrder** table is updated, that the corresponding rows in the **WorkOrderDetail** table are also updated to maintain referential integrity.
What should you do?

A. Use a trigger to enforce referential integrity.
B. Create a `CHECK` constraint on the **WorkOrder** table.
C. Include the `ON UPDATE CASCADE` clause in your `FOREIGN KEY` constraint.
D. Include the `WITH CHECK` clause in your `FOREIGN KEY` constraint.

Answer: C
Section: (none)

Explanation/Reference:
When you include the `ON UPDATE CASCADE` clause, it ensures that if a referenced row in the parent table is updated, then the corresponding child rows are updated and referential integrity is maintained. This would ensure that if a user updated the identifier of a **WorkOrder**, the corresponding rows in the **WorkOrderDetail** table would also be updated.

You should not use a trigger to enforce referential integrity. Triggers should not be used when constraints can accomplish the same task. `FOREIGN KEY` constraints are used to enforce referential integrity.
You should not create a `CHECK` constraint on the **WorkOrder** table. A `CHECK` constraint is used to restrict the data allowed for a column to specific values. A `CHECK` constraint consists of a Boolean expression that evaluates to either `TRUE` or `FALSE`. If the expression evaluates to `TRUE`, the value is allowed for the column, and if the expression evaluates to `FALSE`, the value is not allowed for the column. `CHECK` constraints can be defined at the table level or column level, but only `CHECK` constraints defined at the table level can use columns other than the constrained column in the constraint expression.
You should not include the `WITH CHECK` clause in your `FOREIGN KEY` constraint because this only enables the constraint. By default, `FOREIGN KEY` and `CHECK` constraints are enabled when they are created. You can use the `NOCHECK CONSTRAINT` clause of the `ALTER TABLE` statement to temporarily disable a `FOREIGN KEY` or `CHECK` constraint if you need to insert rows that violate the constraint.

QUESTION 276
You are a database developer. You plan to design a database solution by using SQL Server 2008. A table in a database will store large image files (20-50 MB in size).
You have the following business requirements:
·
The image files are accessible by applications that use Win32 APIs.
·
The image files are part of the database backup.
You need to identify an appropriate strategy to store the image files.
Which strategy should you use?

A. Use an image data type.
B. Use the varbinary(max) data type.
C. Use the varbinary(max) data type along with the FILESTREAM attribute.
D. Store the image file in a file system. Use a varchar data type to store the file location in the database.

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 277
You are a database developer. You plan to design a database solution by using SQL Server 2008. A database contains a table named Employee_Vacation. You are given an updated list of employee vacations used. The list is in an XML formatted file. The extract of the XML format is written in the following manner:

```
<Company Name ="ABC Company Pvt Ltd">
 <EmployeeLeave>
  <Employee ID = "1" Name="Jim Reeves" />
  <Leaves>
   <Leave Date="2008-02-12" />
   <Leave Date="2008-02-13" />
   <Leave Date="2008-02-14" />
  </Leaves>
 </EmployeeLeave>
```

You plan to import the data and update the Employee_Vacation table. You design a query to calculate the number of vacation days used by each employee. You need to ensure that vacation days are accurately counted for each employee. What should you do?

A. Use an XQuery expression along with the LET clause and the count function. Return the count in XMLformat.
B. Use an XML index. Aggregate the number of vacation days for each employee, and then return the totalcount in XML format.
C. Use the OPENXML function to convert XML data into a standard table format.Execute the Transact- SQL count function on the vacation days, and then return the count in XML format.
D. Use an XQuery expression to write the information from XML format to a SQL Server table. Aggregate the number of vacation days from the tables, and then return the count in XML format.

Answer: A
Section: (none)

Explanation/Reference:
XQuery is a language that can query structured or semi-structured XML data. With the xml data type support provided in the Database Engine, documents can be stored in a database and then queried by using XQuery.

QUESTION 278
You are a database developer on an instance of SQL Server 2008. You have two tables in your production database created with the following
Transact-SQL statements:

```
CREATE TABLE ARTrx (
TrxID int PRIMARY KEY,
TrxType varchar(5),
Quantity int,
UnitPrice money,
ExtAmt money,
TaxAmt money,
GrandTotal money,
LoadDate datetime);
CREATE TABLE ARTrxMaster (
TrxID int PRIMARY KEY,
TrxType varchar(5),
Quantity int,
UnitPrice money,
ExtAmt money,
TaxAmt money,
GrandTotal money,
OrigLoadDate datetime,
LoadDate datetime);
```

During the week, data is inserted, deleted, and modified in both tables by different applications. At the end of each week, you need to synchronize
the two tables. You want the synchronization to be performed as follows:
 Any rows that exist in **ARTrx** that do not exist in **ARTrxMaster** should be inserted into the **ARTrxMaster** table if the **TrxType** in **ARTrx** has a
value of 'TX1'.
 All rows that exist in both tables should have all columns in the **ARTrxMaster** table updated to match the values in the **ARTrx** table.
 Any rows that exist in **ARTrxMaster** that do not exist in **ARTrx** and have a **TrxType** value of 'TX1' should be removed from **ARTrxMaster**.
To provide optimum performance, you decide to use a `MERGE` statement to perform the synchronization.
Which actions should you take?

A. Use **ARTrxMaster** as the source table and **ARTrx** as the target table.
   Include a `WHEN MATCHED` clause that includes an `UPDATE`.
   Include a `WHEN NOT MATCHED BY TARGET` clause that includes an `INSERT`.
   Include a `WHEN NOT MATCHED BY SOURCE` clause that includes a `DELETE`.
B. Use **ARTrx** as the source table and **ARTrxMaster** as the target table.
   Include a `WHEN MATCHED` clause that includes an `UPDATE`.
   Include a `WHEN NOT MATCHED BY TARGET` clause that includes an `INSERT`.
   Include a `WHEN NOT MATCHED BY SOURCE` clause that includes a `DELETE`.
C. Use **ARTrx** as the source table and **ARTrxMaster** as the target table.
   Include a `WHEN MATCHED` clause that includes an `UPDATE`.
   Include a `WHEN NOT MATCHED` clause that performs an `INSERT` or a `DELETE` as needed.
D. Use **ARTrxMaster** as the source table and **ARTrx** as the target table.
   Include a `WHEN MATCHED` clause that includes an `UPDATE`.
   Include a `WHEN NOT MATCHED BY TARGET` clause that includes a `DELETE`.
   Include a `WHEN NOT MATCHED BY SOURCE` clause that includes an `INSERT`.

Answer: B
Section: (none)

Explanation/Reference:
The `MERGE` statement allows you to combine the inserts, deletes, and updates, and to use a single statement to perform multiple DML actions.
Using a `MERGE` statement instead of issuing multiple DML statements can improve performance. In a `MERGE` statement, you must specify a source
and a target and include a join. Then, you use the `MATCHED` clauses to specify the actions to be performed. The basic syntax of the `MERGE`
statement is as follows:

```
MERGE [INTO] target_table USING source_table
ON join_condition
[WHEN MATCHED THEN
matched_action]
[WHEN NOT MATCHED [BY TARGET] THEN
notmatched_action]
[WHEN NOT MATCHED BY SOURCE THEN
notmatchedsource_action];
```

The two `WHEN NOT MATCHED THEN` clauses specify the actions to take if the records from the source table are not in the target table, or vice
versa. The `WHEN MATCHED THEN` clause specifies the action to take if the records from the source table are in the target table. When synchronizing
tables, you can use `BY TARGET` or `BY SOURCE` to further control how the synchronization occurs when there are differences in the source data and
the target data. In this scenario, you could use the following `MERGE` statement:

```
MERGE ARTrxMaster AS t
USING ARTrx AS s
ON (t.TrxID=s.TrxID)
WHEN MATCHED THEN
UPDATE SET
t.TrxType = s.TrxType,
t.Quantity = s.Quantity,
t.UnitPrice = s.UnitPrice,
t.ExtAmt = s.ExtAmt,
t.TaxAmt = s.TaxAmt,
t.LoadDate = GETDATE()
WHEN NOT MATCHED BY TARGET AND s.TrxType = 'TX1' THEN
INSERT(TrxID, TrxType, Quantity, UnitPrice, ExtAmt, TaxAmt, LoadDate)
VALUES (s.TrxID, s.TrxType, s.Quantity, s.UnitPrice, s.ExtAmt, s.TaxAmt, GETDATE
())
```

```
WHEN NOT MATCHED BY SOURCE AND t.TrxType = 'TX1' THEN
DELETE
OUTPUT $action, INSERTED.*, DELETED.*;
```

With this statement, the following results would occur under these conditions:
 The `WHEN MATCHED` clause would execute and update the target table (**ARTrxMaster**) if a row existed in both tables. The `UPDATE` statement
does not include a table name because the table to be updated is implicit as the target table in the merge.
 The `WHEN NOT MATCHED BY TARGET` clause would insert rows into the target table (**ARTrxMaster**) if the row does not exist in the source
table and the additional condition specified in the `WHEN` clause is met. Only rows in the **ARTrx** table that have a **TrxType** of 'TX1' would be
inserted into **ARTrxMaster**.
 The `WHEN NOT MATCHED BY SOURCE` clause would delete rows from the target table (**ARTrxMaster**) which do not exist in the source table
(**ARTrx)** if the additional condition specified in the `WHEN` clause is met. Only rows in **ARTrxMaster** with a **TrxType** of 'TX1' would be deleted.
This statement also includes an `OUTPUT` clause. The `OUTPUT` clause allows you to retrieve and display information about the rows affected by the
`MERGE` statement. The `OUTPUT` clause can display this information to the user, insert the data into another permanent or temporary table or table
variable using an `INTO` clause, or pass the data to a nested DML statement for processing. Within the `OUTPUT` clause, you specify the column
values that should be retrieved by using the column names with the `INSERTED` and `DELETED` prefixes. The `DELETED` prefix returns the column
value before the DML operation, and the `INSERTED` prefix returns the column value after the DML operation but before executing any triggers. You
can also use `$action` to return a string indicating which type of DML operation affected the row. In this statement, you specified `$action,`
`INSERTED.*, DELETED.*`. This statement would return a result set of the rows affected by the `MERGE`, with the action that was performed for
each row and the before and after values for each action.
In this scenario, you could automate this synchronization process by using the `MERGE` statement in a job that is scheduled to run weekly.
The options that state you should use **ARTrxMaster** as the source table and **ARTrx** as the target table are incorrect. In this scenario, the table to be
updated when a match is found is the **ARTrxMaster** table. Therefore, it should be specified as the target table, and **ARTrx** should be specified as
the source table.
The option that states you should include a `WHEN NOT MATCHED` clause that performs an `INSERT` or a `DELETE` as needed is incorrect. In this
scenario, you needed to perform different actions depending on the values in stored in each table. Therefore, you must use two `WHEN NOT
MATCHED` clauses, one specifying `BY TARGET` and the other specifying `BY SOURCE`. If you omit the clause, `BY TARGET` is used by default.

QUESTION 279
You are a database developer.
You create a database that uses SQL Server 2008 in an enterprise environment.
You plan to import data from an external source into a table.
You need to ensure that the following tasks are accomplished:

The import is successfully completed even if it encounters rows that fail foreign key constraints.

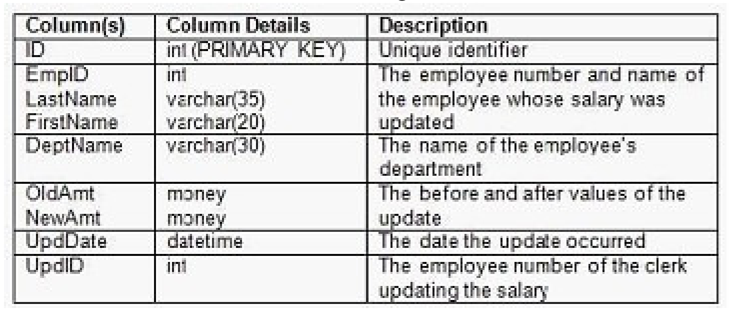
The rows that fail the foreign key constraints during import are inserted into a separate table.
What should you do?

A. Use CHECK constraints.
B. Use an AFTER trigger.
C. Use an INSTEAD OF trigger.
D. Disable the foreign keys during the import process.

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 280
You are a database developer on an instance of SQL Server 2008. You are creating a **SalaryAudit** table to record custom audit information about salary changes for employees.
You have two payroll clerks that are allowed to process salary updates for the twenty departments within your company. At regular intervals, the clerks process salary updates for a large number of employees.
The table will contain the following columns:



Column(s)	Column Details	Description
ID	int (PRIMARY KEY)	Unique identifier

EmpID
LastName
FirstName
	int
	varchar(35)
	varchar(20)
		The employee number and name of the employee whose salary was updated

DeptName	varchar(30)	The name of the employee's department
OldAmt NewAmt	money money	The before and after values of the update
UpdDate	datetime	The date the update occurred
UpdID	int	The employee number of the clerk updating the salary

This audit information will be rarely used, but internal auditors will perform ad hoc queries on the table during company-wide audits that occur every three years. After each audit is complete and the finalized audit reports have been filed, the **SalaryAudit** table will be backed up, and all rows for the audit period will be removed from the table.
You want to minimize storage requirements for the table as much as possible, but support the other requirements with minimal effort.
Which action should you take?

A. Use FILESTREAM storage and implement NTFS compression.
B. Implement column-level compression for the table.
C. Implement page-level compression for the table.
D. Set the `PERSISTED` property for the **EmpID**, **LastName**, **FirstName**, and **UpdID** columns.

Answer: C
Section: (none)

Explanation/Reference:
Page compression is implemented internally by SQL Server to minimize the storage required for duplicate data. SQL Server uses a column-prefix or page-dictionary compression technique to eliminate redundant data within each page. In this scenario, because you have a large amount of duplicate data, you should use page compression. When you implement page compression, SQL Server also implements row compression. Row compression saves storage space for each row by internally compressing each fixed-length numeric, date/time, or character data type. When you implement row compression for a table, columns in the table only use the actual space required, rather than using the number of bytes required for each data type.
You can implement table compression when the table is created by specifying the `DATA_COMPRESSION` option in the `CREATE TABLE` statement. Valid options for the `DATA_COMPRESSION` option are as follows:

`NONE`: Implements no compression for the table.
`ROW`: Implements only row compression for the table.
`PAGE`: Implements row and page compression for the table.

In this scenario, you could create the **SalaryAudit** table with page-level compression using the following statement:

```
CREATE TABLE SalaryAudit (ID int PRIMARY KEY,EmpID int,LastName varchar(35),
FirstName varchar(25),DeptName varchar(30),OldAmt money,NewAmt money,UpdDate
datetime,UpdID int)WITH (DATA_COMPRESSION = PAGE);
```

You can also alter an existing table to use data compression. For example, you could use the following statement to modify a previously created table, **Table1**, to use row compression:

```
ALTER TABLE Table1 REBUILD WITH (DATA_COMPRESSION=ROW);
```

You should note that compression requires additional overhead, especially if the table is frequently accessed or modified, because the data must be compressed and uncompressed when it is accessed or modified. Therefore, you should closely evaluate not only the storage requirements but also how the table is used before implementing compression. In this scenario, the data is written only once to the table when a salary update occurs, and is rarely queried. Therefore, the additional overhead would likely have no adverse effects.
You should not use FILESTREAM storage and implement NTFS compression. FILESTREAM storage is implemented to store large binary objects, such as image or video files, as files on the file system and be able to manage them using Transact-SQL. FILESTREAM data can also be accessed using Win32 APIs. FILESTREAM storage is tightly integrated with most database functionality, including backup and recovery. When you take a database backup, FILESTREAM storage is also backed up unless you override this functionality by performing a partial backup. To create a table that can store FILESTREAM data, you create a table that contains a column of the **varbinary(max)** data type and include the `FILESTREAM` attribute.
You cannot implement column-level compression for the table because column-level compression is not a compression strategy supported by SQL Server. SQL Server can implement compression at the row or page levels only.
You should not set the `PERSISTED` property for the **EmpID**, **LastName**, **FirstName**, and **UpdID** columns. The `PERSISTED` property is only applicable to computed columns, which are not used in this scenario. Computed columns are virtual columns that by default are not physically stored in the table. Each computed column uses the `AS` keyword, followed by an expression that evaluates to a value. The expression can contain constants, functions, operators, and references to other columns within the table. The value of the computed column is calculated each time a query references it executes. Computed columns that specify the `PERSISTED` property are physically stored in the table and recalculated each time a column value referenced in the calculation expression is changed.

QUESTION 281
You maintain a database on an instance of SQL Server 2008. Incoming XML documents are received from several third-party providers and
validated by a client application. After being processed by the client application, the XML documents need to be stored in the database. You also need to create multiple stored procedures that will perform different types of processing on the elements within each XML document.
You want to use minimal development effort to load the data and code the stored procedures. Which action should you take?

A. Use FILESTREAM storage for the XML documents.
B. Create a schema collection before importing the XML documents.
C. Use typed XML when storing the XML documents.
D. Use untyped XML when storing the XML documents.

Answer: D
Section: (none)

Explanation/Reference:
Untyped XML is well-formed XML that is represented by an **xml** data type. Untyped **xml** columns and

variables are defined with the built-in **xml** data type. Untyped XML is not validated in any other way unless you include special code to do so. For example, the following Transact-SQL code creates a table that contains a column, **OrderDetail**, which contains untyped XML:

```
CREATE TABLE MyXMLOrderTable (OrderID int NOT NULL PRIMARY KEY, OrderDetail xml,
LoadDate datetime);
```

Untyped XML is used when you need to load XML that does not conform to its specified schema, you do not have defined schemas, or the XML has already been validated and you do not need to perform validation on the server.

You should not create a schema collection before importing the XML documents. Schema collections are used to validate XML. In this scenario, the XML is validated by a client application. Therefore, this task does not need to be performed on the database server.

You should not use FILESTREAM storage for the XML documents. In this scenario, you wanted to store the XML documents in the database. FILESTREAM storage is used to store data outside the database on the file system.

You should not use typed XML when storing the XML documents because in this scenario, you did not need to perform validation. Typed XML can be used if XML needs to validated, such as if you need to validate that the XML has valid data types and values for specific elements and attributes. To create a typed **xml** column or variable, you must first use the `CREATE XML SCHEMA COLLECTION` statement to create a schema used to validate the XML. Then, you can use the schema collection when declaring an **xml** variable or creating an **xml** column in a table. You can also use an existing XSD when you create the schema collection, if one exists.

QUESTION 282

There are 20 tables in a database. Each one has a primary key but there is no referential integrity. More than 5 Million Rows per table with 10 percent being outdated. A strategy is needed to delete outdate records meeting the following requirements:

- min execution time
- min locking an application execution accessing data
- min development effort

A. use IF with single DELETE statement

B. use single transaction with multiple DELETE statement

C. use while loop with single DELETE statement

D. use multiple TRUNCATE TABLE statements

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 283

You are a database developer on an instance of SQL Server 2008.

Weekly, your finance department receives XML from several branch offices. The data in these XML files must meet certain validation requirements. You currently have an XSD file that you want to use to validate the XML documents. What should you do?

A. Use `OPENXML` including a `WITH` clause.

B. Import the XML documents into an **xml** data type and use the **exist()** method.

C. Use `FOR XML AUTO` with the `XMLSCHEMA` option.

D. Register the XML schema using the `CREATE XML SCHEMA COLLECTION` statement.

Answer: D
Section: (none)

Explanation/Reference:
After the schema is registered, it can be associated with an XML document variable or **xml** data type column, and used to validate the XML. For example, if you previously registered a schema collection named **VendorSchema**, you could use the following statements to declare an **xml** variable or create a table containing an **xml** column with validation:

```
DECLARE @desc xml (VendorSchema)
CREATE TABLE Vendor (ID int PRIMARY KEY,VendorData xml (VendorSchema));
```

You should not use `OPENXML` including a `WITH` clause. `OPENXML` is a rowset provider function that creates a relational view of data contained in an XML document. This function can be used in `SELECT` statements where a table or view would be specified to extract data from an XML document, but cannot be used to validate XML. The `WITH` clause can be used with `OPENXML` to specify a table name that defines the relational structure of the result.
You should not import the XML documents into an **xml** data type and use the **exist()** method. The **exist()** method is an **xml** data type method used to determine if a specific XML node exists. The **exist()** method is not used to validate XML.
You should not use `FOR XML AUTO` with the `XMLSCHEMA` option. The `FOR XML` clause is used with a `SELECT` statement to output relational data in XML format. The `XMLSCHEMA` option allows you to include an inline schema in the generated output.

QUESTION 284
You are a database developer. You plan to design a database solution by using SQL Server 2008.
The database will store multilingual data.
The database will contain a table that has 100 million rows. The table will contain 1,000 columns that are based on the nvarchar(max) data type. For each column, only 2 percent of the rows will be populated.
You need to design the table to optimize storage space.
What should you do?

A. Use row compression.
B. Use NTFS file system compression to reduce the disk space used.
C. Define the columns as sparse columns.
D. Change the column data types to varchar(max).

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 285
You use SQL Server 2008 to design a database that will hold incoming XML responses for an EDI system.
You have the following requirements:
·
The data is accessible to heterogeneous platforms.
·
The database stores various types of reports from multiple sources.
The solution allows search by keywords.
The database stores large amounts of data.
The database is scalable.
You need to design the database to meet the given requirements.
What should you do?

A. Use SQL Server 2008 tables to store data and include proper indexes.

B. Use ANSI text files to store text reports, and use SQL Server 2008 tables to store numerical reports.
C. Save reports in binary format in a file within a Windows folder. Save the path of the file in SQL Server 2008 tables.
D. Store reports in XML format, and use SQL Server 2008 tables to store the data. Index the XML data to improve performance.

Answer: D
Section: (none)

Explanation/Reference:

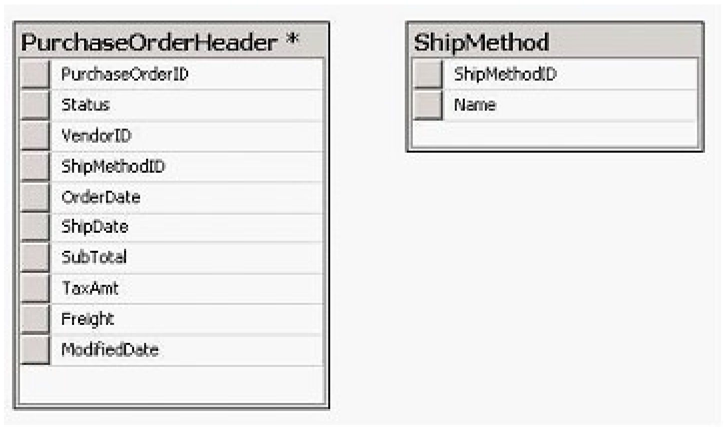
QUESTION 286
You manage a database in an instance of SQL Server 2008. You want to allow users to execute a given query providing different `WHERE` clause values. You want the solution to provide the best performance possible, but also provide maximum security. Which action should you take?

A. Use string concatenation to build the query string and then issue the `EXECUTE` statement, passing it the string.
B. Use a Common Table Expression (CTE).
C. Create a parameterized stored procedure that uses **sp_executesql**.
D. Create a stored procedure that includes the `WITH RECOMPILE` clause to execute the query.

Answer: C
Section: (none)

Explanation/Reference:
In this scenario, you need to dynamically create SQL statements that include different `WHERE` clause values. To provide the best performance, you should use a parameterized stored procedure so that cached execution plans are more likely to be used. When executing dynamic SQL, you should use the **sp_executesql** system stored procedure because it minimizes the likelihood of a SQL injection attack.
You should not use a Common Table Expression (CTE). CTEs are used to make Transact-SQL that includes subqueries more readable, or as an alternative to using a view or temporary table. The subquery can be defined once in a CTE definition and then referenced multiple times in the SQL statement following the CTE definition.
You should not use string concatenation to build the query string and then issue the `EXECUTE` statement, passing it the string. This would increase the likelihood of a SQL injection attack. To minimize the possibility of SQL injection attacks, you should use the **sp_executesql** system stored procedure to execute dynamic SQL instead of using the `EXECUTE` statement.
You should not create a stored procedure that includes the `WITH RECOMPILE` clause to execute the query. The `WITH RECOMPILE` clause is used to force stored procedure compilation. If you include the `WITH RECOMPILE` clause when creating a stored procedure, the stored procedure will be recompiled each time it is called, which may decrease performance.

QUESTION 287
You are a database developer on an instance of SQL Server 2008. Your **Prod** database contains tables that contain purchasing-related data. The database contains the following tables:



You need to create queries against the tables that use dynamic SQL. Your IT management has recently implemented security policies mandating that all Transact-SQL code must minimize the likelihood of SQL injection attacks.
What should you do to minimize the likelihood of SQL injection attacks?

A. Use the `EXECUTE` statement to execute dynamic SQL.
B. Implement all dynamic SQL using CLR functions and procedures.
C. Implement all dynamic SQL within Transact-SQL stored procedures.
D. Use the **sp_executesql** system stored procedure to execute dynamic SQL.

Answer: D
Section: (none)

Explanation/Reference:
SQL injection attacks occur when a user maliciously provides input that is embedded into a dynamic SQL statement. The **sp_executesql** system stored procedure accepts parameters and constructs the dynamic SQL. This eliminates the need to dynamically construct SQL statements with string concatenation, and minimizes the likelihood of SQL injection attacks. This also is more likely to improve performance because with parameters being used, cached execution plans are more likely to be reused. The **sp_executesql** system stored procedure accepts an **@stmt** parameter that contains one or more SQL statements, an optional **@params** parameter to identify parameters used in the SQL statements, and optional user-defined parameter values. The following Transact-SQL illustrates how you might use the **sp_executesql** with the tables in this scenario:
```
-- Selects the names of all shipping methods used for Vendor 84SELECT @sql
=N'SELECT DISTINCT sm.Name ' + N'FROM Purchasing.PurchaseOrderHeader p ' +N'INNER
JOIN Purchasing.ShipMethod sm ' +N'ON p.ShipMethodID = sm.ShipMethodID ' +N'WHERE
p.VendorID = @v';
SELECT @params = N'@v int'EXEC sp_executesql @sql, @params, 84
```
You should not use the `EXECUTE` statement to execute dynamic SQL. Using the `EXECUTE` statement to execute dynamic SQL increases the likelihood of SQL injection attacks.
You should not implement all dynamic SQL using CLR functions and procedures. CLR functions and procedures still introduce the possibility of SQL injection attacks if they pass dynamically constructed SQL to the database for execution.
You should not implement all dynamic SQL within Transact-SQL stored procedures. Although using parameterized stored procedures may decrease the likelihood of SQL injection attacks, SQL injection may still occur when dynamic SQL is executed from within a stored procedure. For example, suppose you have the following stored procedure defined:
```
CREATE PROCEDURE getpos (@sname varchar(50))AS
DECLARE @sql nvarchar(max) =N'SELECT p.PurchaseOrderID, sm.Name ' +N'FROM
PurchaseOrderHeader p ' +N'INNER JOIN ShipMethod sm ' +N'ON p.ShipMethodID = sm.
ShipMethodID ' +
N'WHERE sm.Name LIKE ''' + @sname +N'%'';'
```
With this stored procedure, a user could maliciously pass a parameter of `'''; DROP TABLE ShipMethod;`

`--'` and introduce malicious SQL code.

QUESTION 288
You are a database developer. You plan to design a database solution by using SQL Server 2008. The database will contain a table that will store customer data as XML data. The data supports an application that cannot be altered.
You plan to prevent the following types of errors in the XML data.
NULL values in the Customer Name field
Non-numeric values in the Customer Telephone field.
Invalid values in the Gender field
You need to implement the plan without modifying the application.
What should you do?

A. Use the FileStream data type.
B. Change the XML data type to Typed XML.
C. Use the HierarchyID data type to validate data.
D. Save the XML data in a standard table format. Specify the correct data types, constraints, and NOT NULL parameters in the standard table.

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 289
You are a database developer. You plan to design a database solution by using SQL Server 2008.
A frequently used query takes very long to execute.
You discover that the query frequently uses full-table scans instead of indexes. This causes other queries that modify the table to be blocked.
The indexing strategy on the underlying tables that the query uses can change. You need to design a solution that performs the following tasks:
·
Removes full-table scans
·
Allows the query optimizer to select the appropriate index.
What should you do?

A. Use the INDEX table hint.
B. Use the INDEX(0) table hint.
C. Use the NOEXPAND table hint.
D. Use the FORCESEEK table hint.

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 290
You are a database developer. You plan to design a database solution by using SQL Server 2008.
A database contains a table named Policies. The table contains information about 100 million insurance

policies. A complex stored procedure executes daily to calculate the risk amount of each policy and stores the information in the table. When the stored procedure is executed, users experience poor performance and query time-out errors. The queries used in the stored procedure are optimized for performance. You need to ensure that the disruption to users is minimal while the stored procedure is being executed.
What should you do?

A. Use the READ UNCOMMITTED transaction isolation level.
B. Split the execution of the stored procedure into batches.
C. Write the risk amounts to a table variable before you update the Policies table.
D. Write the risk amounts to a temporary table before you update the Policies table.

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 291
You are a database developer for a retail application. You create a database by using SQL Server 2008 in a distributed enterprise environment that has multiple servers. The same database is implemented on all the servers. The database contains a table that has a surrogate key.
You need to ensure that the following requirements are met:
·
The surrogate key is unique across all servers.
·
The index on the surrogate key is not fragmented because of INSERT operations.

What should you do?

A. Use the timestamp data type.
B. Use the bigint data type. Use the IDENTITY property in the column definition.
C. Use the uniqueidentifier data type. Use the NEWID() function in a default constraint.
D. Use the uniqueidentifier data type. Use the NEWSEQUENTIALID() function in a default constraint.

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 292
You are a database developer. You plan to design a database solution by using SQL Server 2008. You create a table that contains information about Web pages that are added to a Web site. The Web site has a home page and contains various other Web pages. The home page is the root page of the site. All pages except the root page have a link to an upper-level page. The table must support the following design considerations:
- Records of the Web pages that are linked to a particular page can be quickly retrieved.
- The position of a Web page in a collection of linked pages can be quickly retrieved.
- Changing the links to the upper-level pages is a rare requirement.

You need to ensure that the table is designed appropriately. What should you use?

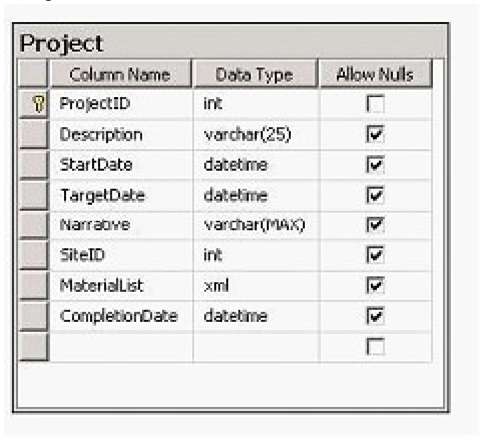
A. Use the XML data type.

B. Use the hierarchyid data type.
C. Use a Parent/Child mechanism that references the same table.
D. Use a Parent/Child mechanism that references one or more additional tables.

Answer: B
Section: (none)

Explanation/Reference:
The hierarchyid data type is a variable length, system data type. Use hierarchyid to represent position in a hierarchy. A column of type hierarchyid does not automatically represent a tree. It is up to the application to generate and assign hierarchyid values in such a way that the desired relationship between rows is reflected in the values.

QUESTION 293
You currently administer a database on an instance of SQL Server 2008. Your **Prod** database contains a **Project** table as shown:



Project

Column Name	Data Type	Allow Nulls
ProjectID	int	☐
Description	varchar(25)	✓
StartDate	datetime	✓
TargetDate	datetime	✓
Narrative	varchar(MAX)	✓
SiteID	int	✓
MaterialList	xml	✓
CompletionDate	datetime	✓
		☐

The **MaterialList** column stores XML documents that represent the materials required to complete each project. The material lists are compiled and automatically transmitted by project managers in the field using a third-party application, and loaded into the database programmatically.
As each material list is added to the **Project** table, you want to ensure that certain elements within the XML conform to the following criteria:
Each material in the material list must have values for the product and stock number elements.
Each material in the material list must have a numeric quantity value.
Each material in the material list must have a value representing the date the item is required at the job site.

Which action should you take?

A. Use typed XML.
B. Use untyped XML.
C. Store the incoming XML in a staging table containing needed constraints.
D. Store the incoming XML into columns in a temporary table to perform validation.

Answer: A
Section: (none)

Explanation/Reference:
Typed XML can be used if XML needs to validated, such as if you need to validate that the XML has valid data types and values for specific elements and attributes. To create a typed **xml** column or variable, you must first use the `CREATE XML SCHEMA COLLECTION` statement to create a schema used to validate the XML. Then, you can use the schema collection when declaring an **xml** variable or creating an **xml** column in a table. You

can also use an existing XSD when you create the schema collection, if one exists.
You should not use untyped XML. Untyped XML is well-formed XML that is represented by an**xml** data type. Untyped **xml** columns and variables are defined with the built-in **xml** data type. Untyped XML is not validated in any other way unless you include special code to do so. Untyped XML is used when you need to load XML that does not conform to a specified schema, you do not have a defined schema, or the XML has already been validated and you do not need to perform validation on the server. In this scenario, you wanted the XML to be validated. Therefore, you should use typed XML.
You should not store the incoming XML into columns in a temporary table to perform validation, or store the incoming XML in a staging table containing needed constraints. In this scenario, you can continue to store the XML in the existing column and perform the required validation. There is no need to create a temporary or staging table that requires intermediate processing.

QUESTION 294
Update statement should be terminated when encounters a shared lock, what is the best solution?

A. use with(holdlock)
B. use sp_who
C. use sp_lock
D. use with(nowait)

Answer: D
Section: (none)

Explanation/Reference:
http://technet.microsoft.com/en-us/library/ms187373.aspx

NOWAIT

Instructs the Database Engine to return a message as soon as a lock is encountered on the table. NOWAIT is equivalent to specifying SET LOCK_TIMEOUT 0 for a specific table.

QUESTION 295
You are a database administrator on an instance of SQL Server 2008. You create two database schemas using the following Transact-SQL statements:
`CREATE SCHEMA Schema1; CREATE SCHEMA Schema2 AUTHORIZATION DevUser;`
You create **User1** with a default schema of **Schema2**. Assuming defaults and no other assigned permissions, which statement(s) about user and roles are true? (Choose all that apply.)

A. **User1** can create tables in **Schema2**.
B. **DevUser** can drop **Schema2** if it contains no database objects.
C. **DevUser** can drop any objects that have been created in **Schema2**.
D. **DevUser** can create database objects in **Schema2**.
E. **DevUser** can copy objects in **Schema2** to **Schema1**.

Answer: BC
Section: (none)

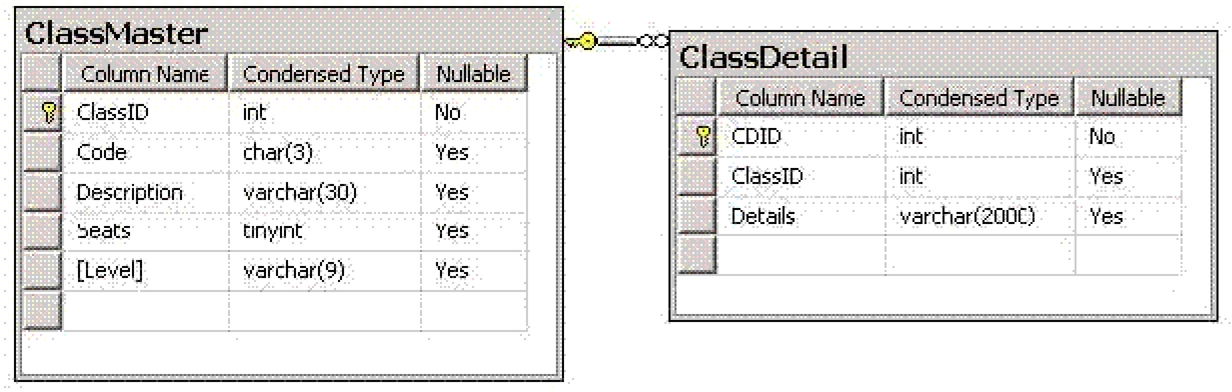
Explanation/Reference:
In this scenario, you created two schemas using `CREATE SCHEMA` statements. Each schema has an owner. The `CREATE SCHEMA` statement can specify an `AUTHORIZATION` clause to identify the schema's owner. The schema's owner may be a user or role. The schema owner automatically has grant permissions on any objects in the schema and has permission to drop the schema if it is empty. By default, the schema owner cannot

create any objects, so additional permissions must be granted to allow this if necessary. However, by default, the schema owner can drop any objects in the schema.
In this scenario, you first created a schema named **Schema1** without specifying a schema owner. If no owner is specified, **dbo** will be the schema's default owner. Next, you created a schema named **Schema2** specifying **DevUser** as the schema's owner. Because **DevUser** was specified as the owner of **Schema2**, **DevUser** can drop **Schema2** if it contains no database objects and can drop any objects created in **Schema2**.
Finally, you created **User1** with a default schema of **Schema2**. When creating or altering a user, you can specify the user's default schema using the `DEFAULT_SCHEMA` clause. If you omit the default schema, **dbo** will be used as the default schema. If the user subsequently creates a database object without specifying a schema, the object will be created in the user's default schema. In this scenario, if **User1** created an object, it would be created in the **Schema2** schema by default. You should note that if the user is authenticated via a Windows group, the user will have no default schema assigned. If such a user creates an object, SQL Server will create a new schema in which the object is created. The new schema created will have the same name as the user that created the object. You should always specify a schema when creating a user.
Schemas contain database objects. Using schemas allows you to manage ownership and permissions on database objects more effectively because the schema and its ownership are separate from the user. Schemas allow you to organize objects and easily grant and revoke permissions on groups of objects. When you grant a user an object permission at the schema level, the permission is automatically given to the user for all objects within the schema. In addition, the permission is also implicitly granted on any future objects created in the schema without any further action. Because users and object ownership are separate, you can drop a user that owns objects without having to first transfer ownership of the objects to another user.
DevUser cannot copy objects in **Schema2** to **Schema1** because **DevUser** is neither the schema owner nor been granted permission at the schema level. In this scenario, you created **Schema1** without specifying an owner, so the owner will default to **dbo**.

QUESTION 296
You manage a database on an instance of SQL Server 2008 at a large educational institution. You have **ClassMaster** and **ClassDetail** tables as shown:
(Click exhibit to view table design)
The **User1** user starts a session and executes the following Transact-SQL:
```
BEGIN TRANSACTION
 UPDATE ClassDetail WITH(TABLOCK)
  SET ClassDetail.Details = ClassDetail.Details + '. This is a Freshman-level
class.'
  FROM ClassMaster INNER JOIN
       ClassDetail ON ClassMaster.ClassID = ClassDetail.ClassID
  WHERE ClassMaster.Level = 'Freshman'
;
```
Then, **User2** starts a session and executes the following Transact-SQL:
```
BEGIN TRANSACTION
 INSERT INTO ClassMaster(Description, Seats, Level)
  VALUES ('Calculus I', 25, 'Sophomore'),
         ('Accounting III', 80, 'Senior'),
         ('World History', 30, 'Freshman');
 DELETE FROM dbo.ClassDetail WHERE CDID = 2;
COMMIT TRANSACTION
```

What is the result?

Exhibit:



ClassMaster

Column Name	Condensed Type	Nullable
ClassID	int	No
Code	char(3)	Yes
Description	varchar(30)	Yes
Seats	tinyint	Yes
[Level]	varchar(9)	Yes

ClassDetail

Column Name	Condensed Type	Nullable
CDID	int	No
ClassID	int	Yes
Details	varchar(2000)	Yes

A. **User1**'s updates are applied first, and then **User2**'s DML operations are performed.
B. **User2**'s session hangs waiting for **User1**'s update to complete.
C. **User1**'s updates are not performed, but **User2**'s DML operations complete successfully.
D. **User2**'s session immediately returns an error.

Answer: B
Section: (none)

Explanation/Reference:
In this scenario, **User1** begins a transaction and executes an `UPDATE` statement, but does not commit the changes. Then, **User2** begins a transaction, but **User1** currently has locks on the table that prevents **User1**'s actions from being performed. **User2**'s session hangs waiting for **User1**'s session to release the locks. SQL Server uses locking to control which transactions have access to which resources at any given time. This prevents one transaction from overwriting another transaction's changes. SQL Server determines the locks that will be acquired based on the type of action being performed. However, you can override default locking behavior if necessary using table hints.
The option that states **User1**'s updates are applied first and then **User2**'s DML operations are performed is incorrect. **User1** does not commit the changes. Therefore, the changes are pending and **User2**'s operations are not performed.
The option that states **User1**'s updates are not performed but **User2**'s DML operations complete successfully is incorrect. In this scenario, **User1**'s updates have not been committed and will remain pending, blocking **User2**'s operations from executing.
The option that states **User2**'s session immediately returns an error is incorrect. **User2**'s session will wait to obtain the necessary locks to perform the required operations.

QUESTION 297
You are a database solutions architect. Your company plans to develop a solution by using a SQL Server 2008 instance.
You design a new database that contains a table to store Microsoft Office documents.
You have the following business requirements:
·
The documents are part of the database backup.
·
The snapshots of the database are used.
You need to use an appropriate data type to store the documents.
Which data type should you use?

A. varchar(max)
B. nvarchar(max)
C. varbinary(max)
D. varbinary(max) by using the FILESTREAM attribute

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 298
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 instance to create a solution.
You create a new database. In the database, a table is used to store Microsoft Office documents.
You require that the snapshots of the database are used and the documents are part of the database backup.
The documents have to be stored by using an suitable data type.

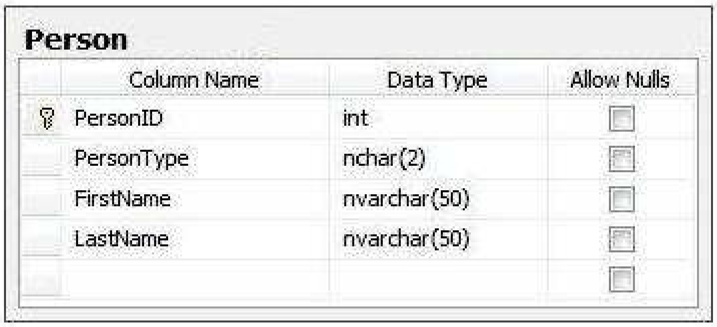
So which data type should you use?

A. varchar(max)
B. varbinary(max)
C. nvarchar(max)
D. varbinary(max) by using the FILESTREAM attribute

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 299
You are a database developer. You plan to design a database solution by using SQL Server 2008. A database contains a table named Person. The structure of the table is as shown in the following exhibit.
(Click the Exhibit button.)


Person

Column Name	Data Type	Allow Nulls
PersonID	int	☐
PersonType	nchar(2)	☐
FirstName	nvarchar(50)	☐
LastName	nvarchar(50)	☐
		☐

The table has the following indexes:
A unique clustered index on the PersonID column named IX_Person_PersonID A nonclustered index on the

FirstName and LastName columns named IX_Person_FirstName_LastName
A nonclustered index on the PersonType column named IX_Person_PersonType that has FirstName and LastName as included columns
The table contains approximately 700,000 records. The approximate number of records for each PersonType is 3,000.
You execute the following query.

```
SELECT P.FirstName, P.LastName
FROM Person P
WHERE P.PersonType = 'DR'
```

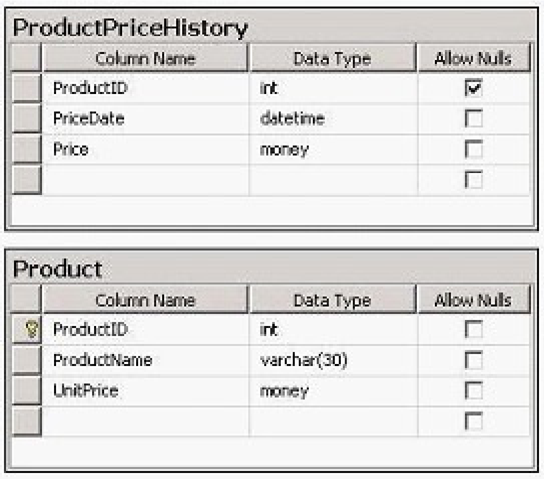
You plan to analyze the performance of the query by using an execution plan. You need to ascertain that the indexes are used optimally.
What should you do?

A. Verify that a clustered index scan operation is performed on the IX_Person_PersonID index.
B. Verify that an index seek operation is performed on the IX_Person_PersonType index.
C. Verify that an index seek operation is performed on the IX_Person_PersonType index, and a key lookup operation is performed on the IX_Person_PersonID index.
D. Verify that an index seek operation is performed on the IX_Person_PersonType index, and an index scan operation is performed on the IX_Person_FirstName_LastName index.

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 300
You are a database developer on an instance of SQL Server 2008. Your database contains the **Product** and **ProductPriceHistory** tables defined as follows:



ProductPriceHistory

Column Name	Data Type	Allow Nulls
ProductID	int	✓
PriceDate	datetime	☐
Price	money	☐
		☐

Product

Column Name	Data Type	Allow Nulls
ProductID	int	☐
ProductName	varchar(30)	☐
UnitPrice	money	☐
		☐

You have a query that references a subquery on the two tables. You decide to use a Common Table Expression (CTE) instead of using a subquery.
Which Transact-SQL successfully implements a CTE?

A.
```
WITH LowPricedProducts (ProductID, ProductName, Price) AS
SELECT * FROM Product;
SELECT * FROM LowPricedProducts ORDER BY ProductName;
(SELECTp.ProductID,p.ProductName,MIN(c.Price)
  FROM Product p INNER JOIN
  ProductPriceHistory c ON c.ProductID = p.ProductID
  GROUP BY p.ProductID, p.ProductName
  HAVING MIN(c.Price) < 10
)
```
B.
```
WITH LowPricedProducts (ProductID, ProductName, Price)
 AS
(SELECTp.ProductID,p.ProductName,MIN(c.Price)
  FROM Product p INNER JOIN
  ProductPriceHistory c ON c.ProductID = p.ProductID
  GROUP BY p.ProductID, p.ProductName
  HAVING MIN(c.Price) < 10
)
SELECT * FROM LowPricedProducts ORDER BY ProductName;
```
C.
```
SELECT p.ProductID, p.ProductName, MIN(c.Price)
 FROM Product p INNER JOIN
      ProductPriceHistory c ON c.ProductID = p.ProductID
 GROUP BY p.ProductID, p.ProductName
 HAVING MIN(c.Price) < 10
 OUTPUT * INTO LowPricedProducts )
SELECT *
 FROM LowPricedProducts
 ORDER BY ProductName;
```
D.
```
WITH LowPricedProducts (ProductID, ProductName, Price) AS
SELECT * FROM LowPricedProducts ORDER BY ProductName;
(SELECTp.ProductID,p.ProductName,MIN(c.Price)
 FROM Product p INNER JOIN
 ProductPriceHistory c ON c.ProductID = p.ProductID
 GROUP BY p.ProductID, p.ProductName
 ORDER BY p.ProductName
 HAVING MIN(c.Price) < 10)
```

Answer: B
Section: (none)

Explanation/Reference:
The following Transact-SQL successfully implements a Common Table Expression (CTE):
```
WITH LowPricedProducts (ProductID, ProductName, Price) AS(SELECTp.ProductID,p.
ProductName,MIN(c.Price)FROM Product pINNER JOIN ProductPriceHistory cON c.
ProductID = p.ProductIDGROUP BY p.ProductID, p.ProductNameHAVING MIN(c.Price) <
10)
SELECT * FROM LowPricedProducts ORDER BY ProductName;
```
When defining a CTE, the `WITH` clause specifies the expression name that will be used in the subsequent statement. The `WITH` clause must contain a column list identifying the available columns, unless all columns in the expression's query have distinct names. The syntax for creating a CTE is as follows:
`WITH expression_name [(column_name [,...n])]AS (CTE_query_definition)`
After you create the CTE, the statement immediately following the CTE definition can reference the CTE expression by name one or more times, as if it were a table or view. Only the columns defined in the CTE expression are accessible.
You can also create two CTEs in a single `WITH` clause by separating the expressions with a comma. Within the `WITH` clause, the CTE can also reference itself or another CTE that you previously created. Note that only one `WITH` clause is allowed, even if the query defining the CTE contains a subquery. In addition, a CTE query definition cannot contain an `ORDER BY`, `COMPUTE`, `COMPUTE BY`, `INTO`, `FOR XML`, or `FOR BROWSE` clause, or an `OPTION` clause that specifies query hints. CTE expressions can be used as an alternative to a view, temporary table, or subquery. Using a CTE expression makes the Transact-SQL code more readable than

using a subquery, especially if the query using the CTE needs to reference the same result set multiple times.
The Transact-SQL that includes two `SELECT` statements after the CTE definition will generate an error because the statement that uses the CTE must be the first statement after the CTE definition. If you executed this code, the following error would occur:

```
Msg 208, Level 16, State 1, Line 16Invalid object name 'LowPricedProducts'.
```

The Transact-SQL that uses an `ORDER BY` clause in the CTE definition will generate an error because an `ORDER BY` clause is not allowed for a query that defines a CTE.
The Transact-SQL that omits the `WITH` keyword and includes an `OUTPUT` clause in the CTE definition will generate an error. The `WITH` keyword must be included to define a CTE. In addition, the `OUTPUT` clause is not allowed for a query that defines a CTE because the `OUTPUT` clause is not valid for `SELECT` statements. However, a DML statement that immediately follows the CTE definition and references the CTE can include an `OUTPUT` clause. When performing inserts, updates, deletes, and merges, you can use the `OUTPUT` clause to obtain and display information about the rows affected by the DML operation. The `OUTPUT` clause can display this information to the user, insert the data into another permanent or temporary table or table variable using an `INTO` clause, or pass the data to a nested DML statement for processing. Within the `OUTPUT` clause, you specify the column values that should be captured by using the column names with the `INSERTED` and `DELETED` prefixes.

QUESTION 301
You are a database developer. You plan to design a database solution by using SQL Server 2008.
A stored procedure uses the INSERT, UPDATE, and DELETE statements separately to load data into a table.
You need to rewrite the stored procedure to use a single statement to load the data.
What should you do?

A. Write a MERGE statement by using a WHEN MATCHED clause and a WHEN NOT MATCHED BY TARGET clause.
B. Write a MERGE statement by using a WHEN MATCHED clause and a WHEN NOT MATCHED BY SOURCE clause.
C. Write a MERGE statement by using a WHEN MATCHED clause, a WHEN NOT MATCHED BY TARGET clause, and a WHEN NOT MATCHED BY SOURCE clause.
D. Write a MERGE statement by using a WHEN MATCHED clause and two WHEN NOT MATCHED BY SOURCE clauses.

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 302
You are a database developer. You plan to design a database solution by using SQL Server 2008. The database will contain a table that has a parent-child relationship to itself. Each child might also be a parent. This might exist up to 10 levels deep. You need to retrieve all levels by using a single Transact-SQL query.
What should you do?

A. Write a query to return the first level, and then add a correlated subquery to get the remaining levels.
B. Write a query to return the first level, and then use the CROSS JOIN operator to join the table back to itself to get the remaining levels.
C. Create a common-table expression to return the first level and then union back to itself to get the remaining levels.
D. Create a view that returns the first level, and then use the FULL OUTER JOIN operator to join the table back to the view to get the remaining levels.

Answer: C
Section: (none)

Explanation/Reference:
A common table expression (CTE) provides the significant advantage of being able to reference itself, thereby creating a recursive CTE. A recursive CTE is one in which an initial CTE is repeatedly executed to return subsets of data until the complete result set is obtained.

A query is referred to as a recursive query when it references a recursive CTE. Returning hierarchical data is a common use of recursive queries, for example: Displaying employees in an organizational chart, or data in a bill of materials scenario in which a parent product has one or more components and those components may, in turn, have subcomponents or may be components of other parents.

QUESTION 303
You are a database developer. You plan to design a database solution by using SQL Server 2008.
The database includes a table that contains the following product inventory information:
·Department
·Class
·Item
·Quantity
You plan to write a query that produces the sum of quantity data broken into the following groups.
·Department
·Department and Class
·Department and Item
·Department, Class, and Item
You need to write the query by using the minimum possible number of Transact-SQL statements.
What should you recommend?

A. Write a single query that contains a GROUP BY clause.
B. Write a single query that contains a GROUP BY WITH CUBE clause.
C. Write a single query that contains a GROUP BY WITH ROLLUP clause.
D. Write a single query that contains a GROUP BY GROUPING SETS clause.

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 304
You are a database developer on an instance of SQL Server 2008. You are creating a tracking application.
You want to create a column in one of your tables that stores GPS latitude and longitude coordinates.
Which data type should you use for the column?

A. xml
B. table
C. geometry
D. geography

Answer: D
Section: (none)

Explanation/Reference:
The **geography** data type represents data in a round-earth coordinate system. The **geography** data type is used to represent data as GPS latitude and longitude coordinates.
You should not use the **xml** data type because this data type is used to store XML data, such as XML documents or XML document fragments.
You should not use the **table** data type because this data type is used to store rows of data from a result set. The **table** data type can be processed later as needed, much like a temporary table. You can also pass table-valued parameters into stored procedures and functions for processing.
You should not use the **geometry** data type. The **geometry** data type represents data in a Euclidean, or flat, coordinate system.

QUESTION 305
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You use SQL Server 2008 to create an online transaction processing (OLTP) database in an enterprise environment. A table named SellingData is contained in the database.

Each record in the table contains data in any one of the following pairs of nullable columns:
- ForeignSellingTargets and ForeignSelling
- InternetSellingTargets and InternetSelling
- ResellerSellingTargets and ResellerSelling.

There're three NOT NULL key columns in the table. A large number of records are inserted on a daily basis into the SellingData table.
From the SellingData table, summary reports are generated, each of which is based on aggregated data from any one of the pairs of nullable columns.

The requirements below must be satisfied:
- The SellingData table cannot be directly modified.
- The performance of the reports is maximized.
- The amount of storage space for each report is minimized.

You have to design a view or views to meet these requirements.
So what should you do?

A. You should create an indexed view from the SellingData table.The table should contain aggregated data of all the columns required by all the reports.
B. In order to make each view contain aggregated data of only the columns required by the respectivereport, you should create multiple indexed views from the SellingData table.
C. In order to make each Report table contain aggregated data of only the columns required by therespective report,you should create multiple Report tables from the SellingData table.And on top of each of the Report tables create views.
D. At the end of each month, you should transfer aggregated new records to a staging table quickly.Create an indexed view from the staging table.The table contains aggregated data of all the columns required by all the reports.

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 306
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You use SQL Server 2008 to create a database solution in an enterprise environment. A table

named PurchaseCustoms which has no indexes contained in your data warehouse. A table named SellRequests is contained in your online transaction processing (OLTP) database. Data between the two tables has to be synchronized on a weekly basis. The following are requirements for the synchronization process:
New records in the SalesOrders table are inserted in the factBuyingHabits table. Records that are deleted from the SalesOrders table are also deleted from the factBuyingHabits table. When a record is modified in the SalesOrders table, the modification is updated in the factBuyingHabits table. You have to design a suitable synchronization solution while using as little coding and administrative efforts as possible. So what should you do?

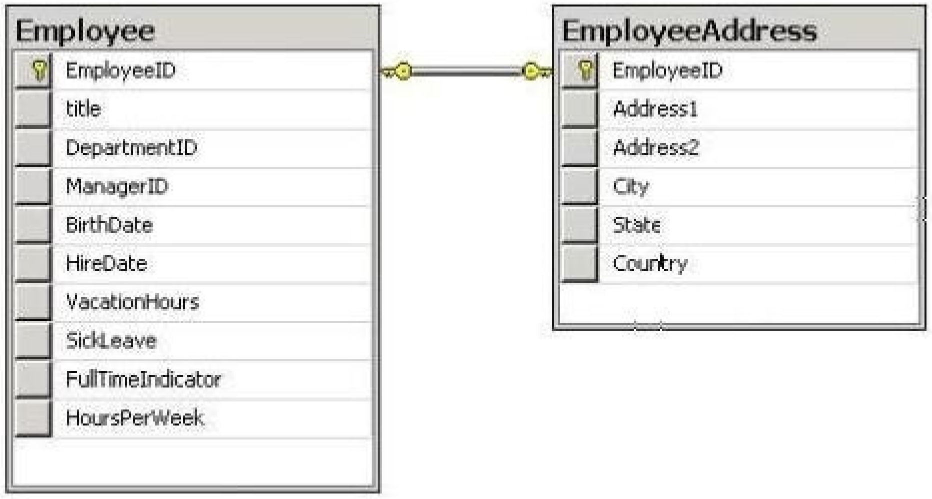
A. You should design a single SSIS package. The package should use the Slowly Changing Dimension task. And then schedule a job to run this package
B. For the INSERT, UPDATE, and DELETE operations each, you should design an SSIS package. And then schedule a job to run this package.
C. For the INSERT, UPDATE, and DELETE operations each, you should write three stored procedures. And then you schedule a job to run the stored procedures in a sequential manner.
D. You should write one stored procedure that contains a MERGE statement to perform the INSERT, UPDATE, and DELETE operations. And then schedule a job to run the stored procedure.

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 307
You work in an International company named TAKEEEN. And you're in charge of the database of your company. There is a SQL Server 2008 instance which hosts a third-party database. A database application is developed for the instance. You have no permissions to modify the database schema.

The database contains two tables that are as shown in the diagram below. On the basis of FullTimeIndicator flag, you intend to extract address information about full-time employees. In order to simplify the extraction process, you have to design a data access layer.



So what should you do?

A. You should design an Entity Data Model.The EMPLOYEES and ADDRESS entities are contained in this Model.
B. On the database, you should create a view.Full-time employees and their address details are included in it.

C. You should re-design the underlying database model.Employee and address information are included in one table.
D. You should design a conceptual Entity Data Model.An entity named EMPLOYEE_ADDRESS is contained in the Model.Make sure that information about employees and their addresses is contained in this entity.

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 308
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. On a server a database is configured to use a common language runtime (CLR). A CLR assembly has to be created. This assembly will be used to make the CLR stored procedure be able to access environment variables available on the server. Besides this, you have to make sure that the CLR assembly has permissions assigned as little as possible. So what should you do?

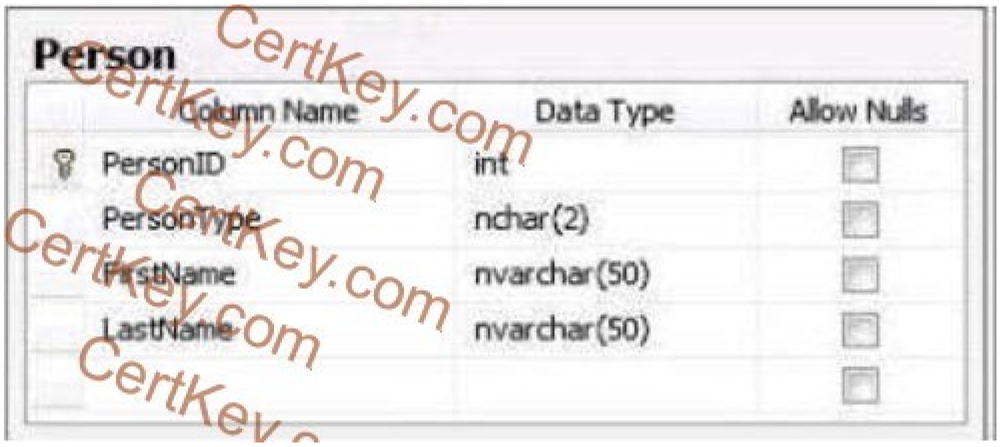
A. You should enable the TRUSTWORTHY database property
B. You should use the SAFE permission set to create the assembly.
C. You should use the UNSAFE permission set to create the assembly
D. You should use the EXTERNAL ACCESS permission set to create the assembly

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 309
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. A table named Person is contained in a database. The exhibit below shows the structure of the table.
The table has these indexes:
A unique clustered index on the PersonID column named IX_Person_PersonID A nonclustered index on the FirstName and LastName columns named IX_Person_FirstName_LastName
A nonclustered index on the PersonType column named IX_Person_PersonType that has FirstName and LastName as included columns There're about 700,000 records in the table. The number of records for each PersonType is about 3,000.
You execute the following query.
SELECT P.FirstName, P.LastName
FROM Person P
WHERE P.PersonType = 'DR'

You have to use an execution plan to analyze the performance of the query. Besides this, you have to make sure that the indexes are used optimally. So what should you do?



Person

	Column Name	Data Type	Allow Nulls
🔑	PersonID	int	☐
	PersonType	nchar(2)	☐
	FirstName	nvarchar(50)	☐
	LastName	nvarchar(50)	☐
			☐

A. You should identify that an index seek operation is performed on the IX_Person_PersonType index.
B. You should identify that a clustered index scan operation is performed on the IX_Person_PersonID index.
C. You should identify that a key lookup operation is performed on the IX_Person_PersonID index and an index seek operation is performed on the IX_Person_PersonType index,
D. You should identify that an index scan operation is performed on the IX_Person_FirstName_LastName index and an index seek operation is performed on the IX_Person_PersonType index.

Answer: A
Section: (none)

Explanation/Reference:

QUESTION 310
A common language runtime (CLR) user-defined scalar function will be contained in the database.
An integer value will be returned by the function.

What should you do to make sure that the computed columns that use the result from this function can be indexed?

A. You should make sure that the logic of the function returns a different value for the same input valuesand the same database state.Make sure that you have set the IsDeterministic property to True.
B. You should make sure that the logic of the function returns the same value for the same input valuesand the same database state.Make sure that you have set the IsDeterministic property to True.
C. You should make sure that a different value is returned by the logic of the function for the same inputvalues and the same database state.Make sure that you have set the IsDeterministic property to False.
D. You should make sure that the same value is returned by the logic of the function returns for the sameinput values and the same database state.Make sure that you have set the IsDeterministic property to False.

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 311

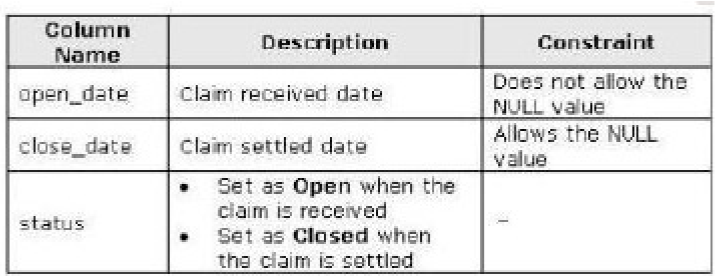
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. The database has a table named Selling which contains 10 million rows.
SELECT s.sale_id, ...
FROM Sales AS s
JOIN Country AS c
ON s.Country_id = c.Country_id
AND c.Country_name = 'USA'
You notice that it takes a long time for the query above to execute. The code segment below shows the summary of the execution plan. |--Hash Match(Inner Join, HASH:([s].[Country_id]) = ([c].[Country_id]) |--Clustered Index Scan(OBJECT:([Country].[PK_Country_Country_id] AS [c]) |--Clustered Index Scan(OBJECT:([Sales].[PK_Sales_Sale_id] AS [s])) What should you do to make sure that the query retrieves data as soon as possible?

A. You should modify the query to use a loop join hint.
B. You should modify the query to use a merge join hint
C. In the Country_id column of the Selling table, you should create a nonclustered index
D. In the Country_name column of the Country table, you should create a nonclustered index.

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 312
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. There's a table named Claims in the database. The following exhibit shows the structure of the Claims table.



Column Name	Description	Constraint
open_date	Claim received date	Does not allow the NULL value
close_date	Claim settled date	Allows the NULL value
status	• Set as **Open** when the claim is received • Set as **Closed** when the claim is settled	–

Only 3% of the claims are open at any point in time.
Queries on claims that have an Open status execute slowly.

So what should you do to make the claim-processing queries perform optimally?

A. You should partition the Claims table on the open_date column by using a partitioning function.
B. You should use a WHERE clause to create an index for the Claims table.Then use the index to include all rows that have a NULL value in the close_date column.
C. You should use a WHERE clause to create a view for the Claims table.Then use the view to include all rows that have a NULL value in the close_date column.

D. You should use a WHERE clause to create a table-valued function for the Claims table.Then use the function to include all rows that have a NULL value in the close_date column.

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 313
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. You intend to design a complex multi-statement stored procedure in the manner below.
CREATE PROCEDURE Sales.GetCustomerActivity
@StartDate datetime
AS
SELECT order_id, order_date, customer_id
FROM Sales.Orders
WHERE order_date >= @StartDate
...
You notice that sometimes the stored procedure executes slowly and this is caused by the first statement in the stored procedure. So what should you do to make sure that the stored procedure is always executed normally in the minimum possible time?

A. You should run the EXEC sp_recompile GetCustomerActivity command
B. You should add the WITH RECOMPILE clause to modify the stored procedure
C. A plan guide should be created to apply the OPTION (RECOMPILE) clause to the first statement.
D. The first statement in the stored procedure should be replaced with the following Transact- SQL statement. UPDATE STATISTICS Sales.GetCustomerActivity WITH RESAMPLE Certkey.com - Make You Succeed To Pass IT Exams
Certkey 70-451

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 314
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database. There's a table which is partitioned into four geographic regions in the database. The table is named Selling.Items.
Look at the stored procedures below:
CREATE STORED PROCEDURE usp_Update
@RegionID tinyint
AS
UPDATE Sales.Inventory
SET Qty = T.CurrentQuantity
FROM Sales.Inventory I
JOIN Sales.TempData T ON I.ItemID = T.ItemID
AND I.RegionID = @RegionID;

You use the stored procedure above, the UPDATE statement locks the Sales.Inventory table.

So what should you do to prevent this from happening?

A. You should run this Transact-SQL statement: ALTER TABLE Sales.Inventory SETLOCK_ESCALATION = AUTO
B. You should run this Transact-SQL statement: ALTER TABLE Sales.Inventory SETLOCK_ESCALATION = TABLE
C. The usp_Update stored procedure should be modified to use the NOLOCK table hint for the UPDATEstatement
D. The usp_Update stored procedure should be modified to use the SERIALIZABLE table hint for theUPDATE statement

Answer: A
Section: (none)

Explanation/Reference:

QUESTION 315
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You create a database solution by using SQL Server 2008. The database will hold incoming XML responses for an EDI system.
You have the requirements for the database:

- The database is scalable and accessible to heterogeneous platforms.
- The database stores large amounts of data and various types of reports from multiple sources.
- The solution allows search by keywords.

So what should you do to make the database to meet these requirements?

A. You should store data and include proper indexes by using SQL Server 2008 tables.
B. You should store numerical reports by using SQL Server 2008 tables and store text reports by usingANSI text files.
C. You should save reports in binary format in a file within a Windows folder.Save the path of the file in SQL Server 2008 tables.
D. You should store reports in XML format, and store the data by using SQL Server 2008 tables.Index the XML data to improve performance.

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 316
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. A stored procedure load data into a table by using the INSERT, UPDATE, and DELETE statements separately. Now you want to load the data by using a single statement. Therefore, the stored procedure has to be rewritten. So what should you do?

A. You should use a WHEN MATCHED clause and a WHEN NOT MATCHED BY SOURCE clause to write a MERGE statement
B. You should use a WHEN MATCHED clause and a WHEN NOT MATCHED BY TARGET clause to write a MERGE statement.

C. You should use a WHEN MATCHED clause and two WHEN NOT MATCHED BY SOURCE clauses to write a MERGE statement by

D. You should use a WHEN MATCHED clause, a WHEN NOT MATCHED BY TARGET clause, and a WHEN NOT MATCHED BY SOURCE clause to write a MERGE statement

Answer: D
Section: (none)

Explanation/Reference:

QUESTION 317
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. You own the Sales schema and a user named MarketingMan is the owner of the Marketing schema.
You are allowed to create objects in all schemas in the database.
But users of the Marketing schema are not allowed to access the Sales schema which has a table named Clients.
In the Marketing schema, you intend to create a stored procedure for the marketing team.
The stored procedure will select data from the Clients table and will be owned by MarketingMan.

What should you do to make sure that the marketing team is able to execute the stored procedure?

A. You should use the EXECUTE AS OWNER option to create the procedure.

B. You should use the EXECUTE AS SELF option to create the procedure.

C. You should use the EXECUTE AS CALLER option to create the procedure.

D. You should use the EXECUTE AS USER=MarketingUser option to create the procedure.

Answer: C
Section: (none)

Explanation/Reference:

QUESTION 318
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution.The database will contain a table.

Customer data are stored in the table as XML data.
The data supports an application that cannot be altered.
You plan to prevent the types of errors below in the XML data:

- Invalid values in the Gender field,
- NULL values in the Customer Name field,
- Non-numeric values in the Customer Telephone field.

What should you do to implement the plan while not modifying the application?

A. You should use the FileStream data type.

B. You should change the XML data type to Typed XML.

C. You should validate data by using the HierarchyID data type.

D. You should save the XML data in a standard table format.Specify the correct data types, constraints, and NOT NULL parameters in the standard table.

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 319
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. You are creating a SQL Agent job. This SQL Agent job updates data in two related databases on two different servers by using Transact-SQL.
You require that each evening the job can only execute once and it uses transactions to ensure that all updates are rolled back if an error occurs;
the databases on each server use the full-recovery model;
transaction log backups for the two databases occur at different times.

You have to make sure that when a database is restored on either server, the two databases are restored to a state, the state should reflect the last time the job successfully executed.

So what should you do?

A. You should use the NO_WAIT termination clause to make sure that both databases are altered.
B. You should use a marked transaction to restore both databases when a database failure occurs.
C. You should use a saved transaction to restore both databases when a database failure occurs.
D. You should make sure that the two databases always keep pace with each other by using the WindowsSync Manager.

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 320
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution. A database contains a table named Clauses. Information about 100 million insurance clauses are stored in this table. A complex stored procedure executes daily to count the risk amount of each clause and stores the information in the table. Query time-out errors occur and users experience poor performance when the stored procedure executes. In order to optimize the performance, you optimize the queries used in the stored procedure. You must make sure that when the stored procedure is being executed, the users will be affected as little as possible and the disruption to them is minimal. So what should you do?

A. You should use the READ UNCOMMITTED transaction isolation level
B. You should split the execution of the stored procedure into batches
C. Before you update the Clauses table, the risk amounts should be written to a temporary table
D. Before you update the Clauses table, the risk amounts should be written to a table variable

Answer: B
Section: (none)

Explanation/Reference:

QUESTION 321
You work in an International company named TAKEEEN. And you're in charge of the database of your company. You intend to use SQL Server 2008 to create a database solution.The product inventory information such as Item, Class, Quantity, Department is contained in a table of the database. You intend to write a query by using as little number of Transact-SQL statements as possible. The query will produce the sum of quantity data broken into these groups:
Department; Department and Item; Department and Class; Department, Class, and Item.
What should you do to write the query?

A. You should write a single query and the query should include a GROUP BY clause
B. You should write a single query that the query should include a GROUP BY WITH CUBE clause.
C. You should write a single query that the query should include a GROUP BY WITH ROLLUP clause
D. You should write a single query that the query should include a GROUP BY GROUPING SETS clause

Answer: D
Section: (none)

Explanation/Reference: