Q. 1
You are a database developer for A Datum Corporation. You are creating a database that will store statistics for 15 different high school sports. This information will be used by 50 companies that publish sports information on their web sites. Each company's web site arranges and displays the statistics in a different format.

You need to package the data for delivery to the companies. What should you do?

A. Extract the data by using SELECT statements that include the FOR XML clause.
B. Use the **sp_makewebtask** system stored procedure to generate HTML from the data returned by SELECT statements.
C. Create Data Transformation Services packages that export the data from the database and place the data into tab-delimited text files.
D. Create an application that uses SQL_DMO to extract the data from the database and transform the data into standard electronic data interchange (EDI) files.

Answer: A.
Explanation: The data will be published at the company's web site. XML is a markup language for documents containing structured information. XML is well suited to provide rich web documents. SQL queries can return results as XML rather than standard rowsets. These queries can be executed directly or from within stored procedures. To retrieve results directly, the FOR XML clause of the SELECT statement is used. Within the FOR XML clause an XML mode can be specified. These XML modes are RAW, AUTO, or EXPLICIT.

Incorrect answers:
B: The sp_makeweb stored procedure is used to return results in HTML format rather than as standard rowsets. XM
is a more sophisticated format than HTML and is therefore preferred in this situation.
C: A tab-delimited file can be analyzed in any spreadsheet supporting tab-delimited files, such as Microsoft Excel
This format isn't suitable for web sites, however.
D: SQL-DMO is not used for creating data that can be published on web sites.
Note: SQL-DMO is short for SQL Distributed Management Objects and encapsulates the objects found in SQL Server 2000 databases. It allows applications written in languages that support Automation or COM to administer all parts of a SQL Server installation; i.e., it is used to create applications that can perform administrative duties.

Q. 2
You are a database developer for a mail order company. The company has two SQL Server 2000 computers named CORP1 and CORP2. CORP1 is the online transaction processing server. CORP2 stores historical sales data. CORP2 has been added as a linked server to CORP1.

The manager of the sales department asks you to create a list of customers who have purchased floppy disks. This list will be generated each month for promotional mailings. Floppy disks are represented in the database with a category ID of 21.

You must retrieve this information from a table named SalesHistory. This table is located in the Archive database, which resides on CORP2. You need to execute this query from CORP1.

Which script should you use?

A.
```
EXEC sp_addlinkedserver ‘CORP2’, ‘SQL Server’
GO
SELECT CustomerID FROM CORP2.Archive.dbo.SalesHistory
WHERE CategoryID = 21
```

B.
```
SELECT CustomerID FROM OPENROWSET (‘SQLOLEDB’, ‘CORP2’; ‘p*word’, ‘SELECT
CustomerID FROM Archive.dbo.SalesHistory WHERE CategoryID = 21’)
```

C.
```
SELECT CustomerID FROM CORP2.Archive.dbo.SalesHistory
WHERE CategoryID = 21
```

D.
```
EXEC sp_addserver ‘CORP2’
GO
SELECT CustomerID FROM CORP2.Archive.dbo.SalesHistory
WHERE CategoryID = 21
```

Answer: C.
Explanation: A simple SELECT FROM statement with a WHERE clause is required in the scenario. Usually the code would be written as:

```
SELECT CustomerID
FROM SalesHistory
WHERE CategoryID = 21
```

However the SalesHistory table is located on another server. This server has already been set up as a linked server so we are able to directly execute the distributed query. We must use a four-part name consisting of:

1. Name of server
2. Name of database
3. DBO
4. Name of table

In this scenario it is: CORP2.Archive.dbo.SalesHistory

Note: sp_linkedserver
To set up a linked server, the sp_linkedserver command can be used. Syntax:

```
sp_addlinkedserver [ @server = ] 'server'
   [ , [ @srvproduct = ] 'product_name' ]
   [ , [ @provider = ] 'provider_name' ]
   [ , [ @datasrc = ] 'data_source' ]
   [ , [ @location = ] 'location' ]
   [ , [ @provstr = ] 'provider_string' ]
   [ , [ @catalog = ] 'catalog' ]
```

Incorrect answers:
A: This linked server has already been set up. We don’t have to set it up.
B: OPENROWSET is not used to access linked servers. The OPENROWSET method is an alternative to accessing tables in a linked server and is a one-time, ad hoc method of connecting and accessing remote data using OLE DB.
D: sp_addserver is not a valid stored procedure name.

Q. 3
You are a database developer for Trey Research. You create two transactions to support the data entry of employee information into the company's database. One transaction inserts employee name and address information into the database. This transaction is important. The other transaction inserts employee demographics information into the database. This transaction is less important.

The database administrator has notified you that the database server occasionally encounters errors during periods of high usage. Each time this occurs, the database server randomly terminates one of the transactions.

You must ensure that when the database server terminates one of these transactions, it never terminates the more important transaction. What should you do?

A. Set the DEADLOCK_PRIORITY to LOW for the transaction that inserts the employee name and address information.
B. Set the DEADLOCK_PRIORITY to LOW for the transaction that inserts the employee demographics information.
C. Add conditional code that checks for server error 1205 for the transaction that inserts the employee name and address information. If this error is encountered, restart the transaction.
D. Add the ROWLOCK optimizer hint to the data manipulation SQL statements within the transactions
E. Set the transaction isolation level to SERIALIZABLE for the transaction that inserts the employee name and address information.

Answer: B.
Explanation: We have a deadlock problem at hand. Transactions are randomly terminated.

We have two types of transactions:

the important transaction that inserts employee name and address information
the less important transaction that inserts employee demographic information

The requirement is that when the database server terminates of these transactions, it never terminates the more important transaction.

By setting the DEADLOCK_PRIORITY to LOW for the less important transaction, the less important transaction will be the preferred deadlock victim. When a deadlock between an important and a less important transaction occurs the less important would always be the preferred deadlock victim and terminated. A more important transaction would never be terminated.

We cannot expect only two transactions running at the same time. There could be many less important transactions and many important transactions running at the same time.

We could imagine that two important transactions become deadlocked. In that case, one of them would be the chosen deadlock victim and terminated. But the requirement was that in a deadlock situation the more important transaction would never be terminated, and in this case both are equally important.

Note: Deadlocks
In SQL Server 2000, a single user session may have one or more threads running on its behalf. Each thread may acquire or wait to acquire a variety of resources, such as locks, parallel query execution-related resources, threads, and memory. With the exception of memory, all these resources participate in the SQL Server deadlock detection scheme. Deadlock situations arise when two processes have data locked, and each process cannot release its lock until other processes have released theirs. Deadlock detection is performed by a separate thread called the lock monitor thread. When the lock monitor initiates a deadlock search for a particular thread, it identifies the resource on which the thread is waiting. The lock monitor then finds the owner for that particular resource and recursively continues the deadlock search for those threads until it finds a cycle. A cycle identified
in this manner forms a deadlock. After a deadlock is identified, SQL Server ends the deadlock by automatically choosing the thread that can break the deadlock. The chosen thread is called the deadlock victim. SQL Server
rolls back the deadlock victim's transaction, notifies the thread's application by returning error message number 1205, cancels the thread's current request, and then allows the transactions of the non-breaking threads to continue. Usually, SQL Server chooses the thread running the transaction that is least expensive to undo as the deadlock victim. Alternatively, a user can set the DEADLOCK_PRIORITY of a session to LOW. If a session's setting is set to LOW, that session becomes the preferred deadlock victim. Since the transaction that inserts employee demographics information into the database is less important than the transaction that inserts employee name and address information the DEADLOCK_PRIORITY of the transaction that inserts employee demographics information should be set to LOW.

Incorrect answers:

A: If a session's setting is set to LOW, that session becomes the preferred deadlock victim. Since the transaction that inserts employee name and address information into the database is more important than the transaction that inserts employee demographics information, the DEADLOCK_PRIORITY of the transaction that inserts employee name and address information should not be set to LOW.

C: Error 1205 is returned when a transaction becomes the deadlock victim. Adding conditional code to the transaction that inserts the employee name and address information to check for this error, and specifying that the transaction should restart if this error is encountered, would cause the transaction to restart. This would ensure that an important transaction would never be terminated, which was the requirement. There is a drawback with this proposed solution though: it is inefficient and performance would not be good. It would be better to lower the DEADLOCK_PRIORITY of the less important transactions.

D: ROWLOCK optimizer hint is a table hint that uses row-level locks instead of the coarser-grained page- and table-level locks.

E: Choosing the highest transaction level would increase the number of locks. This could not ensure that certain transactions (the ones with high priority, for example) would never be locked.

Note: When locking is used as the concurrency control method, concurrency problems are reduced, as this allows all transactions to run in complete isolation of one another, although more than one transaction can be running at any time. SQL Server 2000 supports the following isolation levels:

Read Uncommitted, which is the lowest level, where transactions are isolated only enough to ensure that physically corrupt data is not read;
Read Committed, which is the SQL Server 2000 default level;
Repeatable Read; and
Serializable, which is the highest level of isolation.

Where high levels of concurrent access to a database are required, the optimistic concurrent control method should be used.

Q. 4

You are a database developer for your company's SQL Server 2000 online transaction processing database. Many of the tables have 1 million or more rows. All tables have a clustered index. The heavily accessed tables have at least one non-clustered index. Two RAID arrays on the database server will be used to contain the data files. You want to place the tables and indexes to ensure optimal I/O performance.

You create one filegroup on each RAID array. What should you do next?

A. Place tables that are frequently joined together on the same filegroup.
Place heavily accessed tables and all indexes belonging to those tables on different filegroups.

B. Place tables that are frequently joined together on the same filegroup.
Place heavily accessed tables and the nonclustered indexes belonging to those tables on the same filegroup.

C. Place tables that are frequently joined together on different filegroups.
Place heavily accessed tables and the nonclustered indexes belonging to those tables on different filegroups.

D. Place tables that are frequently joined together on different filegroups.
Place heavily accessed tables and the nonclustered indexes belonging to those tables on the same filegroup.

Answer: C.
Explanation: Database performance can be improved by placing heavily accessed tables in one filegroup and placing the table's nonclustered indexes in a different filegroup on different physical disk arrays. This will improve performance because it allows separate threads to access the tables and indexes. A table and its clustered index cannot be separated into different filegroups as the clustered index determines the physical order of the data in the table. Placing tables that are frequently joined together on different filegroups on different physical disk arrays can also improve database performance. In addition, creating as many files as there are physical disk arrays so that there is one file per disk array will improve performance because a separate thread is created for each file on each disk array in order to read the table's data in parallel.

Log files and the data files should also, if possible, be placed on distinct physical disk arrays.

Incorrect Answers:

A: Placing tables that are frequently joined together on the same filegroup will not improve performance, as it minimizesthe use of multiple read/write heads spread across multiple hard disks and consequently does not allow parallel queries. Furthermore, only nonclustered indexes can reside on a different file group to that of the table.

B: Placing tables that are frequently joined together on the same filegroup will not improve performance, as it minimizes the use of multiple read/write heads spread across multiple hard disks and consequently does not allow parallel queries.

D: Placing heavily accessed tables and the nonclustered indexes belonging to those tables on the same filegroup will not improve performance. Performance gains can be realized by placing heavily accessed tables and the nonclustered indexes belonging to those tables on different filegroups on different physical disk arrays. This will improve performance because allow separate threads to access the tables and indexes.

Q. 5
You are a database developer for your company's SQL Server 2000 database. You update several stored procedures in the database that create new end-of-month reports for the sales department. The stored procedures contain complex queries that retrieve data from three or more tables. All tables in the database have at least one index.

Users have reported that the new end-of-month reports are running much slower than the previous version of the reports. You want to improve the performance of the reports.

What should you do?

A. Create a script that contains the Data Definition Language of each stored procedure. Use this script as a workload file for the Index Tuning Wizard.

B. Capture the execution of each stored procedure in a SQL Profiler trace. Use the trace file as a workload file for the Index Tuning Wizard.

C. Update the index statistics for the tables used in the stored procedures.

D. Execute each stored procedure in SQL Query Analyzer, and use the **Show Execution Plan** option.

E. Execute each stored procedure in SQL Query Analyzer, and use the **Show Server Trace** option.

Answer: E.
Explanation: Several stored procedures have been updated. The stored procedures contain complex queries. The performance of the new stored procedures is worse than the old stored procedures.

We use Show trace option of SQL Query Analyzer in order to analyze and tune the stored procedures. The Show Server Trace command provides access to information used to determine the server-side impact of a query.

Note: The new Show Server Trace option of the Query Analyzer can be used to help performance tune queries, stored procedures, or Transact-SQL scripts. What it does is display the communications sent from the Query Analyzer (acting as a SQL Server client) to SQL Server. This is the same type of information that is captured by the SQL Server 2000 Profiler.

Note 2: The Index Tuning Wizard can be used to select and create an optimal set of indexes and statistics for a SQL Server 2000 database without requiring an expert understanding of the structure of the database, the workload, or the internals of SQL Server. To build a recommendation of the optimal set of indexes that should be in place, the wizard requires a workload. A workload consists of an SQL script or an SQL Profiler trace saved to a file or table containing SQL batch or remote procedure call event classes and the Event Class and Text data columns. If an existing workload for the Index Tuning Wizard to analyze does not exist, one can be created using SQL Profiler. The report output type can be specified in the Reports dialog box to be saved to a tab-delimited text file.

Reference: BOL, Analyzing Queries

Incorrect answers:

A: The Index Tuning Wizard must use a workload, produced by an execution of SQL statements, as input. The Index Tuning Wizard cannot use the code of stored procedures as input.
Note: The SQL language has two main divisions: Data Definition Language, which is used to define and manage all the objects in an SQL database, and the Data Manipulation Language, which is used to select, insert, delete or alter tables or views in a database. The Data Definition Language cannot be used as workload for the Index Tuning Wizard.

B: Tuning the indexes could improve the performance of the stored procedures. However, no data has changed and the queries are complex. We should instead analyze the server-side impact of a query by using the Show Server Trace command.

C: The selection of the right indexes for a database and its workload is complex, time-consuming, and error-prone even for moderately complex databases and workloads. It would be better to use the Index Tuning Wizard if you want to tune the indexes.

D: The execution plan could give some clue how well each stored procedure would perform. An Execution Plan describes how the Query Optimizer plans to, or actually optimized, a particular query. This information is useful because it can be used to help optimize the performance of the query. However, the execution plan is not the best method to analyze complex queries.

Q. 6
You are a database developer for wide world importers. You are creating a database that will store order information. Orders will be entered in a client/server application. Each time a new order is entered, a unique order number must be assigned. Order numbers must be assigned in ascending order. An average of 10, 000 orders will be entered each day.

You create a new table named Orders and add an OrderNumber column to this table. What should you do next?

A. Set the data type of the column to **uniqueidentifier**.

B. Set the data type of the column to **int**, and set the IDENTITY property for the column.

C. Set the data type of the column to **int**.
Create a user-defined function that selects the maximum order number in the table.

D. Set the data type of the column to **int**.
Create a **NextKey** table, and add a **NextOrder** column to the table.
Set the data type of the **NextOrder** column to int.
Create a stored procedure to retrieve and update the value held in the **NextKey**.

Answer: B.
Explanation: In MS SQL Server 2000, identifier columns can be implemented by using the IDENTITY property which allows the database designer to specify an identity number for the first row inserted into the table and an increment to be added to successive identity numbers. When inserting values into a table with an identifier column, MS SQL Server 2000 automatically generates the next identity value by adding the increment to the previous identity value. A table can have only one column defined with the IDENTITY property, and that column must be defined using the decimal, int, numeric, smallint, bigint, or tinyint data type. The default increment value by which the identity number grows is 1. Thus identity values are assigned in ascending order by default.

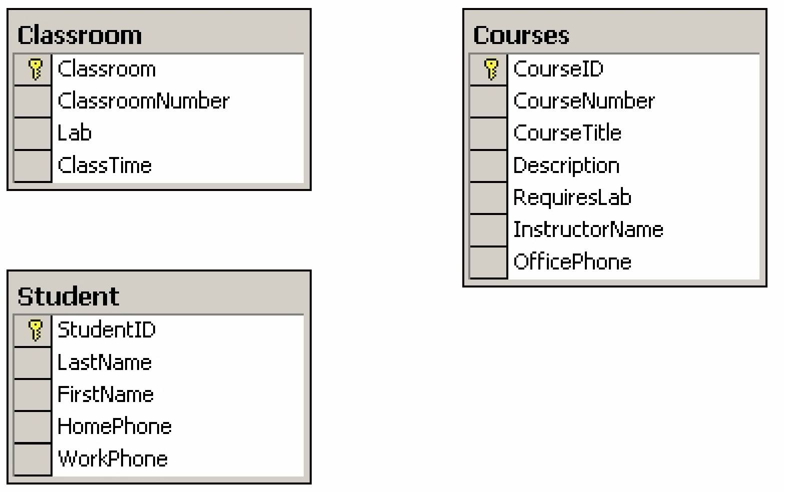
Incorrect answers:

A: MS SQL Server 2000 uniqueidentifier is used during table replication. In this process a unique column for each row in the table being replicated is identified. This allows the row to be identified uniquely across multiple copies of the table.

C: Functions are subroutines that encapsulate frequently performed logic. Any code that must perform the logic incorporated in a function can call the function rather than having to repeat all of the function logic. SQL Server 2000 supports two types of functions: built-in functions and user-defined functions. There are two types of user-defined functions: scalar user-defined functions, which return a scalar value, and inline user-defined functions, which return a table.

D: The creation of additional tables to track order number is inappropriate in this scenario. It would require cascading FOREIGN KEY constraints with the OrderNumber column in the Orders table, which would require manual updating before the OrderNumber column in the Orders table could be updated automatically.

Q. 7
You are a database developer for a technical training center. Currently, administrative employees keep records of students, instructors, courses, and classroom assignments only on paper. The training center wants to eliminate the use of paper to keep records by developing a database to record this information. You design the tables for this database. Your design is shown in the exhibit.



You want to promote quick response times for queries and minimize redundant data. What should you do?

A. Create a new table named **Instructors**.
Include an **InstructorID** column, and **InstructorName** column, and an **OfficePhone** column. Add an **InstructorID** column to the **Courses** table.

B. Move all the columns from the **Classroom** table to the **Courses** table, and drop the **Classroom** table.

C. Remove the PRIMARY KEY constraint from the **Courses** table, and replace the PRIMARY KEY constraint with a composite PRIMARY KEY constraint based on the **CourseID** and **CourseTitle**.

D. Remove the **ClassroomID** column, and base the PRIMARY KEY constraint on the **ClassroomNumber** and **ClassTime** columns.

Answer: A.
Explanation: A normalized database is often the most efficient. This database design is not normalized. The data on the instructors are contained in the Courses table. This would duplicate information whenever an Instructor has more than one course; InstructorName and OfficePhone would have to be registered for every course.

We normalize the database in the following steps:

Create a new table called Instructors.
Create a new column in the Instructors table called InstructorID. This is the given candidate for Primary key.

Add the InstructorName and OfficePhone columns to the Courses table.
Remove the InstructorName and Office Phone columns from the Courses table (not in scenario).
Add the InstructorID column to the Courses table. This column will later be used to create a foreign key constraint to the InstructorID column of the Instructors table.

Incorrect answers:

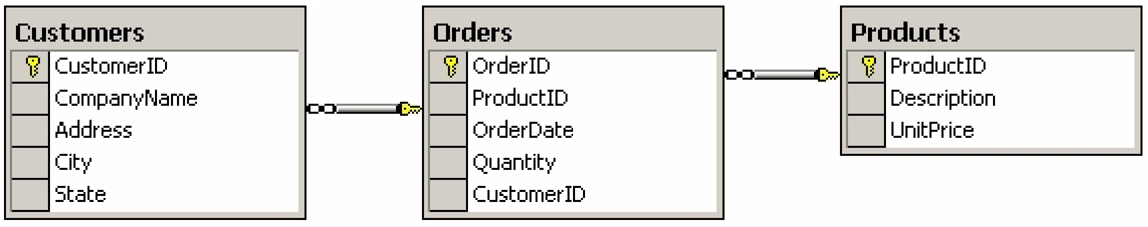
B: Moving all columns from the Classroom table to the Courses table would only make matters worse. Every student's data would have to be entered for every course that student took. We would have an even more denormalized database.

C: By removing the Primary Key constraint on the CourseID of the Courses table and replacing it with a composite Primary Key constraint on the CourseID and CourseTitle columns would make the database more denormalized. It would not allow two courses with the same CourseTitle, so every semester (or year) the school would have to invent new names for the courses.

D: Changing the Primary Key constraint on the Classroom table would not improve the situation; on the contrary, the ClassroomID column would be redundant.

This procedure doesn't address the problem with the InstructorName and OfficePhone columns in the Courses table.

Q. 8
You are designing a database that will contain customer orders. Customers will be able to order multiple products each time they place an order. You review the database design, which is shown in the exhibit.



You want to promote quick response times for queries and minimize redundant data. What should you do? (Each correct answer presents part of the solution. Choose two.)

A. Create a new order table named **OrderDetail**.
Add **OrderID**, **ProductID**, and **Quantity** columns to this table.

B. Create a composite PRIMARY KEY constraint on the **OrderID** and **ProductID** columns of the **Orders** table.

C. Remove the **ProductID** and **Quantity** columns from the **Orders** table.

D. Create a UNIQUE constraint on the **OrderID** column of the **Orders** table.

E. Move the **UnitPrice** column from the **Products** table to the **Orders** table.

Answer: A, C.
Explanation: From a logical database design viewpoint we see that there is some problem with the relationship between the Orders tables and the Products table. We want to have the following relationship between those two tables:

Every order contains one or several products.
Every product can be included in 0, 1, or several orders.

In short, we want a many-to-many relationship between the Orders and Products table, but SQL Server doesn't allow many-to-many relationship, so we have to implement the many-to-many relation via two one-to-many relations by using an extra table that will connect the Orders and the Products table. We do this as follows:

Create a new table OrderDetail.
Add the OrderID, ProductID, and Quantity columns to the OrderDetail table.
Remove the Quantity and the ProductID columns from the Orders table.
Create a foreign key constraint on the OrderID column in the OrderDetail table referencing the OrderID column in the Orders table.
Create a foreign key constraint on the ProductID column in the OrderDetail table referencing the ProductID column in the Products table.

We have now normalized the database design and the benefits are faster query response time and removal of redundant data.

Another less theoretical line of thought is the realization that the OrderID, ProductID and Quantity columns would be of primary concern in the transaction, thus it would be beneficial to create a new table that contains these columns and to remove the Quantity column from the Order table to reduce redundant data.

Incorrect answers:
B: Making a composite primary key out of the OrderID and ProductID columns of the Orders table is not a good idea. From a logical database design standpoint the ProductID doesn't restrict the non-key columns of the Orders table at all, and it should not be part of the Primary Key. Instead, the Orders table should be split into two tables.

D: Creating a UNIQUE constraint on the OrderID column of the Orders table ensures that the values entered in the OrderID column are unique and would prevent the use of null values. It doesn't, however, address the problem of the relationship between the Orders and Products table, which have to be adjusted.

E: Moving the UnitPrice column from the Products table to the Orders table would be counterproductive. The UnitPrice column stores the price of a product and belongs to the Products table and shouldn't be moved to the Orders table. The only way to fix the problem with the Products and Orders table is to add a new table to connect them.

Q. 9
You are the database developer for a publishing company. You create the following stored procedure to report the year-to-date sales for a particular book title:

```
CREATE PROCEDURE get_sales_for_title
%title varchar(80), @ytd_sales int OUTPUT
AS
SELECT @ytd_sales = ytd_sales
FROM titles
WHERE title = @title
IF @@ROWCOUNT = 0
        RETURN(-1)
ELSE
        RETURN(0)
```

You are creating a script that will execute this stored procedure. If the stored procedure executes successfully, it should report the year-to-date sales for the book title. If the stored procedure fails to execute, it should report the following message:
“No Sales Found”

How should you create the script?

A.
```
DECLARE @retval int
DECLARE @ytd int
EXEC get_sales_for_title ‘Net Etiquette’, @ytd IF
@retval < 0
      PRINT ‘No sales found’
ELSE
      PRINT ‘Year to date sales: ’ + STR (@ytd)
GO
```

B.
```
DECLARE @retval int
DECLARE @ytd int
EXEC get_sales_for_title ‘Net Etiquette’, @ytd OUTPUT IF
@retval < 0
      PRINT ‘No sales found’
ELSE
      PRINT ‘Year to date sales: ’ + STR (@ytd)
GO
```

C.
```
DECLARE @retval int
DECLARE @ytd int
EXEC get_sales_for_title ‘Net Etiquette’,@retval OUTPUT IF
@retval < 0
        PRINT ‘No sales found’
ELSE
        PRINT ‘Year to date sales: ’ + STR (@ytd)
GO
```

D.
```
DECLARE @retval int
DECLARE @ytd int
EXEC @retval = get_sales_for_title ‘Net Etiquette’, @ytd OUTPUT IF
@retval < 0
        PRINT ‘No sales found’
ELSE
        PRINT ‘Year to date sales: ’ + STR (@ytd)
GO
```

Answer: D.
Explanation: The stored procedure that reports the year-to-date sales for a particular book title is a RETURN procedure. We must save the return code when the stored procedure is executed so that the return code value in the stored procedure can be used outside the procedure. In this example, @retval is the return code and is DECLARED in line 1; the stored procedure is ‘get_sales_for_title’; and ‘Net Etiquette’ in a book title.
The correct syntax for a RETURN procedure is:

```
DECLARE return code
EXEC return code = stored procedure OUTPUT
```

This example has an additional ytd, or YearToDate variable, that is DECLARED in line 2. In this example the correct syntax should be:

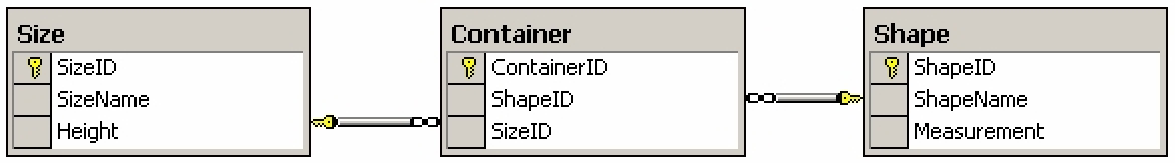
```
DECLARE @retval int
DECLARE @ytd
EXEC @retval = get_sales_for_title ‘Net Etiquette’, @ytd
OUTPUT
```

Incorrect answers:
A: The syntax in line 3 of this code executes the stored procedure without first saving the return code.

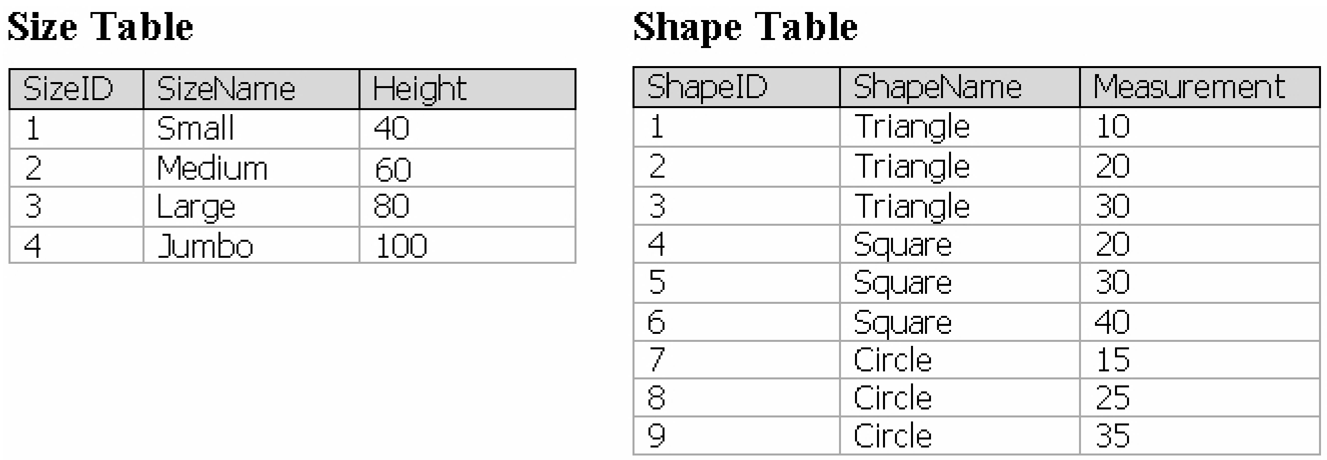
B: The syntax in line 3 of this code executes the stored procedure without first saving the return code.

C: The syntax in line 3 of this code executes the stored procedure without first saving the return code.

Q. 10
You are a database developer for a container manufacturing company. The containers produced by your company are a number of different sizes and shapes. The tables that store the container information are shown in the Size, Container, and Shape Tables exhibit.



A sample of the data stored in the tables is shown in the Sample Data exhibit.



Size Table

SizeID	SizeName	Height
1	Small	40
2	Medium	60
3	Large	80
4	Jumbo	100

Shape Table

ShapeID	ShapeName	Measurement
1	Triangle	10
2	Triangle	20
3	Triangle	30
4	Square	20
5	Square	30
6	Square	40
7	Circle	15
8	Circle	25
9	Circle	35

Periodically, the dimensions of the containers change. Frequently, the database users require the volume of a container. The volume of a container is calculated based on information in the shape and size tables.

You need to hide the details of the calculation so that the volume can be easily accessed in a SELECT query with the rest of the container information. What should you do?

A. Create a user-defined function that requires **ContainerID** as an argument and returns the volume of the container.

B. Create a stored procedure that requires **ContainerID** as an argument and returns the volume of the container.

C. Add a column named volume to the **Container** table. Create a trigger that calculates and store the volume in this column when a new container is inserted into the table.

D. Add a computed column to the **Container** table that calculates the volume of the container.

Answer: A.
Explanation: Calculated columns can be placed directly into SELECT statements. Here we want to hide the details of the calculation, though. We hide the calculation by defining a scalar user-defined function that does the calculation.

Note 1: *User defined functions are a new feature of SQL Server 2000.*
Functions are subroutines that are made up of one or more Transact-SQL statements that can be used to encapsulate code for reuse. User-defined functions are created using the CREATE FUNCTION statement, modified using the ALTER FUNCTION statement, and removed using the DROP FUNCTION statement. SQL Server 2000 supports two types of user-defined functions: scalar functions, which return a single data value of the type defined in a RETURN clause, and table-valued functions, which return a table. There are two types of table-valued functions: inline, and multi-statement.

Note 2: *On computed columns*
A computed column is a virtual column that is computed from an expression using other columns in the same table and is not physically stored in the table. The expression can be a non-computed column name, constant, function, variable, and any combination of these connected by one or more operators but cannot be a subquery. Computed columns can be used in SELECT lists, WHERE clauses, ORDER BY clauses, or any other locations in which regular expressions can be used. However, a computed column cannot be used as a DEFAULT or FOREIGN KEY constraint definition or with a NOT NULL constraint definition but it can be used as a key column in an index or as part of any PRIMARY KEY or UNIQUE constraint if the computed column value is defined by a deterministic expression and the data type of the result is allowed in index columns.

Incorrect answers:
B: A return value of a stored procedure cannot be used in the SELECT list of a query.

Note: SQL Server 2000 stored procedures can return data as output parameters, which can return either data or a cursor variable; as codes, which are always an integer value; as a result set for each SELECT statement contained in the stored procedure or any other stored procedures called by the stored procedure; and as a global cursor that can be referenced outside the stored procedure. Stored procedures assist in achieving a consistent implementation of logic across applications. The SQL statements and logic needed to perform a commonly performed task can be designed, coded, and tested once in a stored procedure. Each application needing to perform that task can then simply execute the stored procedure. Coding business logic into a single stored procedure also offers a single point of control for ensuring that business rules are correctly enforced.

Stored procedures can also improve performance. Many tasks are implemented as a series of SQL statements. Conditional logic applied to the results of the first SQL statements determines which subsequent SQL statements are executed. If these SQL statements and conditional logic are written into a stored procedure, they become part of a single execution plan on the server. The results do not have to be returned to the client to have the conditional logic applied; all of the work is done on the server.

C: Only using an Insert Trigger would not work. The value would not be updated when the dimension of the container changes. An Update Trigger would be required as well.

Note: Triggers are a special type of stored procedure and execute automatically when an UPDATE, INSERT, or DELETE statement is issued against a table or view. Triggers can also be used to automatically enforce business rules when data is modified and can be implemented to extend the integrity-checking logic of constraints, defaults, and rules. Constraints and defaults should be used whenever they provide the required functionality of an application. Triggers can be used to perform calculations and return results only when UPDATE, INSERT or DELETE statements are issued against a table or view. Triggers return the result set generated by any SELECT statements in the trigger. Including SELECT statements in triggers, other than statements that only fill parameters, is not recommended because users do not expect to see any result sets returned by an UPDATE, INSERT, or DELETE statement.

D: SQL Server tables can contain computed columns. Computed columns can only use constants, functions, and other columns in the same table. A computed column cannot use a column of another table. We cannot use a computed column to store the size of a container.

Q. 11
You are a database developer for a hospital. There are four supply rooms on each floor of the hospital, and the hospital has 26 floors. You are designing an inventory control database for disposable equipment. Certain disposable items must be kept stored at all times. As each item is used, a barcode is scanned to reduce the inventory count in the database. The supply manager should be paged as soon as a supply room has less than the minimum quantity of an item.

What should you do?

A. Create a stored procedure that will be called to update the inventory table. If the resultant quantity is less than the restocking quantity, use the **xp_logevent** system stored procedure to page the supply manager.

B. Create an INSTEAD OF UPDATE trigger on the inventory table. If the quantity in the **inserted** table is less than the restocking quantity, use SQLAgentMail to send an e-mail message to the supply manager's pager.

C. Create a FOR UPDATE trigger on the inventory table. If the quantity in the **inserted** table is less than the restocking quantity, use the xp_sendmail system stored procedure to page the supply manager.

D. Schedule the SQL server job to run at four-hour intervals.
Configure the job to use the **@notify_level_page = 2** argument.
Configure the job so that it tests each item's quantity against the restocking quantity.
Configure the job so that it returns a false value if the item requires restocking.
This will trigger the paging of the supply manager.