Microsoft 070-229

Designing and Implementing Databases with Microsoft SQL Server 2000 Enterprise Edition

Version 3.1

Important Note
Please Read Carefully

Study Tips
This product will provide you questions and answers along with detailed explanations carefully compiled and written by our experts. Try to understand the concepts behind the questions instead of cramming the questions. Go through the entire document at least twice so that you make sure that you are not missing anything.

Further Material
For this test TestKing also provides:
* Study Guide. Concepts and labs.
* Interactive Test Engine Examinator. Check out an Examinator Demo at
http://www.testking.com/index.cfm?pageid=724

Latest Version
We are constantly reviewing our products. New material is added and old material is revised. Free updates are available for 90 days after the purchase. You should check your member zone at TestKing an update 3-4 days before the scheduled exam date.

Here is the procedure to get the latest version:

1. Go to www.testking.com
2. Click on **Member zone/Log in**
3. The latest versions of all purchased products are downloadable from here. Just click the links.

For most updates, it is enough just to print the new questions at the end of the new version, not the whole document.

Feedback
Feedback on specific questions should be send to feedback@testking.com. You should state: Exam number and version, question number, and login ID.

Our experts will answer your mail promptly.

Copyright

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. XML 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.

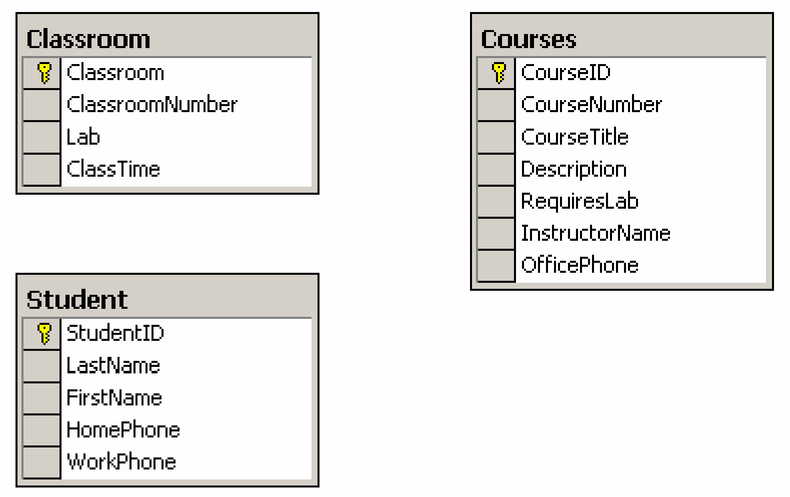
Incorrect answers:

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.

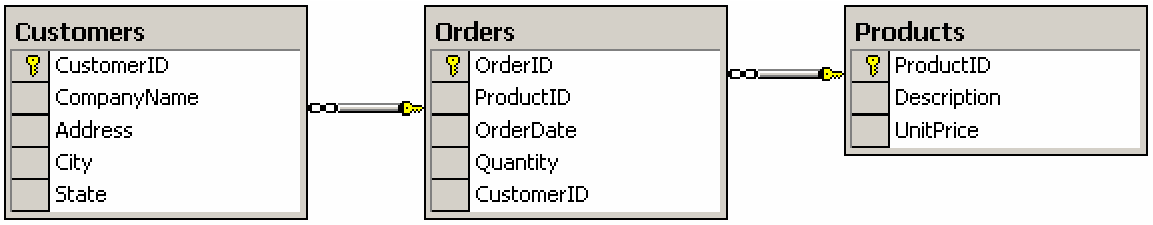
C: By removing the Primary Key constraint on the CourseID of the Courses table and replacing it with a composite Primary Key constraint on the CourseID and CourseTitle columns would make the database more denormalized. It would not allow two courses with the same CourseTitle, so every semester (or year)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

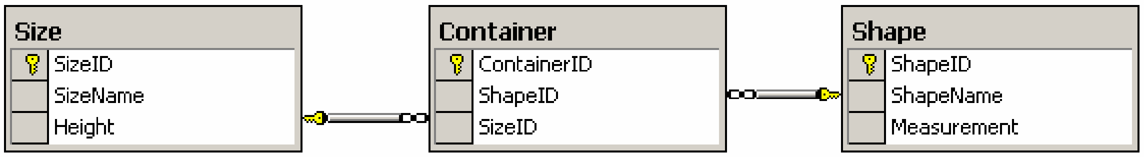
```
DECLARE @retval int
DECLARE @ytd
EXEC @retval = get_sales_for_title ‘Net Etiquette’, @ytd
OUTPUT
```

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

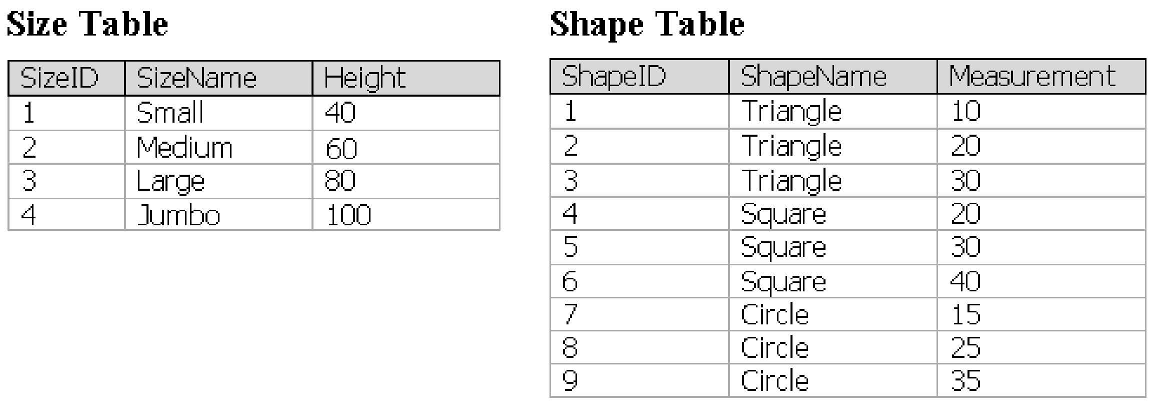
B: The syntax in line 3 of this code executes the stored procedure without first saving the return code.

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.

Answer: C.
Explanation: A FOR UPDATE trigger can be used to check the data values supplied in INSERT and UPDATE statements and to send an e-mail message once the data value reaches a certain value. xp_sendmail is the function used in MS SQL 2000 for send messages to a defined recipient.

Incorrect answers:
A: xp_logevent logs a user-defined message in the MS SQL Server log file and in the Windows 2000 Event Viewer. This solution does not meet the requirements of this scenario.

B: An INSTEAD OF UPDATE trigger can be used to check the data values supplied in INSERT and UPDATE statements and is used in place of the regular action of the UPDATE statement. SQLAgentMail can be configured to send an e-mail message when an alert is triggered or when a scheduled task succeeds or fails. Thus the INSTEAD OF UPDATE trigger can be used to generate an alert that SQLAgentMail can be configured to respond to, but the INSTEAD OF UPDATE trigger replaces the normal update procedure with the send alert procedure; in other words, it would send the alert without updating the table and would thus compromise data integrity.

D: The supply manager should be paged as soon as a condition has been met, i.e. when the supply room has less than the minimum quantity of an item. Scheduling the SQL server job to run at four-hour intervals will not page the supply manager as soon as the condition is met. Instead, the supply manager will be paged only when the scheduled SQL server job is run, which could be up to 4 hours after the condition has been met. Thus, this solution does not meet the requirements of this scenario.

Q. 12
You are the developer of a database that supports time reporting for your company. Usually there is an average of five users accessing this database at one time, and query response times are less than one second. However, on Friday afternoons and Monday mornings, when most employees enter their timesheet data, the database usage increases to an average of 50 users at one time. During these times, the query response times increase to an average of 15 to 20 seconds.

You need to find the source of the slow query response times and correct the problem. What should you do?

A. Use the **sp_lock** and **sp_who** system stored procedures to find locked resources and to identify processes that are holding locks.
Use this information to identify and redesign the transactions that are causing the locks.

B. Query the **sysprocesses** and **sysobjects** system tables to find deadlocked resources and to identify which processes are accessing those resources.
Set a shorter lock timeout for the processes that are accessing the deadlock resources.

C. Query the **sysprocesses** system table to find which resources are being accessed.
Add clustered indexes on the primary keys of all of the tables that are being accessed.

D. Use the **sp_monitor** system stored procedure to identify which processes are being affected by the increased query response times.
Set a less restrictive transaction isolation level for these processes.

Answer: A.
Explanation: One possible and likely cause of the long query response time during peak hours is that resources are being locked. The system stored procedure sp_lock can be used to return a result set that contains information about resources that are locked, and sp_who provides information about current SQL Server 2000 users and processes. The information returned by sp_who can be filtered to return only those processes that are not idle. This makes it possible to identify which resources are being locked and which processes are responsible for creating those locks.

Incorrect answers:
B: The sysprocesses table holds information about processes running on SQL Server 2000. These processes can be client processes or system processes, while sysobjects contains information on all database objects such as tables, views, triggers, stored procedures, etc. Using these tables is not the best way to find locks.

C: The sysprocesses system table holds information about client and system processes running on SQL Server 2000. A clustered index is particularly efficient on columns that are often searched for a range of values. Once the row with the first value is found using the clustered index, rows with subsequent indexed values are guaranteed to be physically adjacent to it. However, clustered indexes are not a good choice for columns that undergo frequent changes, as this results in the entire row moving because SQL Server must keep the data values of a row in a clustered index in physical order. This is an important consideration in high-volume transaction processing systems where data tends to be volatile. In this scenario, large amounts of data are being entered into the table; hence, a clustered index on the table would hamper database performance.

D: sp_monitor is used to keep track, through a series of functions, of how much work has done by SQL Server. Executing sp_monitor displays the current values returned by these functions and shows how much they have changed since the last time the sp_monitor was run. sp_monitor is not the correct stored procedure to use in detecting locked resources.

Q. 13
You are a database developer for an insurance company. The insurance company has a multi-tier application that is used to enter data about its policies and the owners of the policies. The policy owner information is stored in a table named Owners. The script that was used to create this table is shown in the exhibit.

```
CREATE TABLE Owners
          (
          OwnerID int IDENTITY (1, 1) NOT NULL,
          FirstName char(20) NULL,
          LastName char(30) NULL,
          BirthDate date NULL,
          CONSTRAINT PK_Owners PRIMARY KEY (Owner ID)
          )
```

When information about policy owners is entered, the owner's birth date is not included; the database needs to produce a customized error message that can be displayed by the data entry application. You need to design a way for the database to validate that the birth date is supplied and to produce the error message if it is not.

What should you do?

A. Add a CHECK constraint on the **BirthDate** column.

B. Create a rule, and bind the rule to the **BirthDate** column.

C. Alter the **Owners** table so that the **BirthDate** column does not allow null.

D. Create a trigger on the Owners table that validates the BirthDate column.

Answer: D.
Explanation: 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. In this scenario a trigger is required, as a CHECK constraint cannot return a custom error message.

Incorrect Answers
A: CHECK constraints should be used instead of triggers when they meet the functionality of the application. In this scenario, CHECK constraints do not meet the functionality of the application, as CHECK constraints do not allow the generation of customized error messages.
B: Rules are used in cases where backward compatibility is required and perform the same function as CHECK constraints. CHECK constraints are the preferred over rules, as they are also more concise than rules. CHECK constraints however do not allow the generation of customized error messages.
C: Altering the Owners table so that the BirthDate column is defined NOT NULL will prevent the entry of null values but will not generate a customized error message, thus it does not meet the requirements of this scenario.

Q. 14
You are the database developer for a large brewery. Information about each of the brewery's plants and the equipment located at each plant is stored in a database named Equipment. The plant information is stored in a table named Location, and the equipment information is stored in a table named Parts. The scripts that were used to create these tables are shown in the Location and Parts Scripts exhibit.

```
CREATE TABLE Location
          (
          LocationID int NOT NULL,
          LocationName char (30) NOT NULL UNIQUE,
          CONSTRAINT PK_Location PRIMARY KEY (LocationID)
          )
CREATE TABLE Parts
          (
          PartID int NOT NULL,
          LocationID int NOT NULL,
          PartName char (30) NOT NULL,
          CONSTRAINT PK_Parts PRIMARY KEY (PartID),
          CONSTRAINT FK_PartsLocation FOREIGN KEY (Location ID)
               REFERENCES Location (LocationID)
          )
```

The brewery is in the process of closing several existing plants and opening several new plants. When a plant is closed, the information about the plant and all of the equipment at that plant must be deleted from the database. You have created a stored procedure to perform this operation. The stored procedure is shown in the Script for sp_DeleteLocation exhibit.

```
CREATE PROCEDURE sp_DeleteLocation @LocName char(30) AS
BEGIN
     DECLARE @PartID int
     DECLARE crs_Parts CURSOR FOR
          SELECT p.PartID
          FROM Parts AS p INNER JOIN Location AS l
               ON p.LocationID = @LocName
          WHERE l.LocationName = @LocName
     OPEN crs_Parts
     FETCH NEXT FROM crs_Parts INTO @PartID
     WHILE (@@FETCH_STATUS <> -1)
     BEGIN
          DELETE Parts WHERE CURRENT OF crs_Parts
          FETCH NEXT FROM crs_Parts INTO @PartID
     END
     CLOSE crs_Parts
     DEALLOCATE crs_Parts
     DELETE Location WHERE LocationName = @LocName
END
```

This procedure is taking longer than expected to execute. You need to reduce the execution time of the procedure.

What should you do?

A. Add the WITH RECOMPILE option to the procedure definition.

B. Replace the cursor operation with a single DELETE statement.

C. Add a BEGIN TRAN statement to the beginning of the procedure, and add a COMMIT TRAN statement to the end of the procedure.

D. Set the transaction isolation level to READ UNCOMMITTED for the procedure.

E. Add a nonclustered index on the **PartID** column of the **Parts** table.

Answer: B.
Explanation: Cursors are useful for scrolling through the rows of a result set. However they require more overhead and should be avoided if other solutions are possible.

There is a foreign key constraint between the two tables, and rows must be deleted from both tables. A simple DELETE statement cannot accomplish this. Two DELETE statements must be used; first one DELETE statement on the Parts table and then a DELETE statement on the Location table. This is the best solution.

By replacing the cursor operation with a single DELETE statement on the parts table in the code above we would get an optimal solution consisting of only two DELETE statements.

Incorrect answers:
A: Specifying the WITH RECOMPILE option in the stored procedure definition indicates that SQL Server should not cache a plan for this stored procedure, so the stored procedure is recompiled each time it is executed. The WITH RECOMPILE option should only be used when stored procedures take parameters whose values differ widely between executions of the stored procedure, resulting in different execution plans to be created each time. Use of this option causes the stored procedure to execute more slowly because the stored procedure must be recompiled each time it is executed.

C: A transaction is a sequence of operations performed as a single logical unit of work. Programmers are responsible for starting and ending transactions at points that enforce the logical consistency of the data. This can be achieved with BEGIN TRANSACTION, COMMIT TRANSACTION and ROLLBACK TRANSACTION. BEGIN TRANSACTION represents a point at which the data referenced by a connection is logically and physically consistent. If errors are encountered, all data modifications made after the BEGIN TRANSACTION can be rolled back to return the data to this known state of consistency. Each transaction lasts until either it completes without errors and COMMIT TRANSACTION is issued to make the modifications a permanent part of the database, or errors are encountered and all modifications are erased with a ROLLBACK TRANSACTION statement. Thus the three statements must be used as a unit. This solution does not make provision for the ROLLBACK TRANSACTION.

D: The isolation property is one of the four properties a logical unit of work must display to qualify as a transaction. It is the ability to shield transactions from the effects of updates performed by other concurrent transactions. In this scenario, concurrent access to the rows in question is not an issue as these rows pertain to plants that have been shut down. Thus, no updates or inserts would be performed by other users.

E: By adding a nonclustered index on the PartID column of the Parts table, the JOIN statement would execute more quickly and the execution time of the procedure would decrease. A greater gain in performance would be achieved by replacing the cursor operation with a single DELETE statement, though.

Q. 15
You are a database developer for an insurance company. Information about the company's insurance policies is stored in a SQL Server 2000 database. You create a table named Policy for this database by using the script shown in the exhibit.

```
CREATE TABLE Policy
          (
          PolicyNumber int NOT NULL DEFAULT (0),
          InsuredLastName char (30) NOT NULL,
          InsuredFirstName char (20) NOT NULL,
          InsuredBirthDate datetime NOT NULL,
          PolicyDate datetime NOT NULL,
          FaceAmount money NOT NULL,
          CONSTRAINT PK_Policy PRIMARY KEY (PolicyNumber)
          )
```

Each time the company sells a new policy, the policy must be assigned a unique policy number. The database must assign a new policy number when a new policy is entered.

What should you do?

A. Create an INSTEAD OF INSERT trigger to generate a new policy number, and include the policy number in the data inserted into the table.

B. Create an INSTEAD OF UPDATE trigger to generate a new policy number, and include the policy number in the data inserted into the table.

C. Create an AFTER UPDATE trigger to generate a new policy number, and include the policy number in the data inserted into the table.

D. Replace the DEFAULT constraint with an AFTER INSERT trigger that generates a new policy number and includes the policy number in the data inserted into the table.

Answer: A.
Explanation: This scenario requires a constraint that ensures the uniqueness of the data entered into reach row of a particular column. This can be achieved by means of triggers. An INSTEAD OF INSERT trigger can be used to replace of the regular action of the INSERT statement. In this scenario, it can replace the data being inserted with a unique numeric value.

Incorrect answers:

B: An INSTEAD OF UPDATE trigger can be used to replace of the regular action of the UPDATE statement on a table or a view. However, when a new policy is sold, new data will be inserted rather than updated in the table. Thus, the trigger will not fire, as no UPDATE is applied to the table. An INSTEAD OF INSERT trigger should be used instead.

C: An AFTER UPDATE trigger can be used to execute a function after an UPDATE is applied to a table. However, when a new policy is sold, new data will be inserted rather than updated in the table. Thus, the trigger will not fire, as no UPDATE is applied to the table. Furthermore, AFTER triggers are not executed if a constraint violation occurs, as it is executed after the constraint is checked. Thus, AFTER triggers cannot be used for the prevention of constraint violations as in this scenario.

D: AFTER triggers are not executed if a constraint violation occurs, as it is executed after the constraint is checked. If we remove the DEFAULT statement on the PolicyNumber column, the NOT NULL constraint will prevent the AFTER INSERT trigger from firing when a null value is entered in the column. Thus AFTER triggers cannot be used for the prevention of constraint violations as in this scenario.

Q. 16
You are a member of a database development team for a telecommunications company. Another developer on the team, Marc, has created a table named Customers in the Corporate database. Because the table contains confidential information, he has granted SELECT permissions on the table only to the other members of your team.

You are developing an application that will allow employees in the marketing department to view some of the information in the Customers table. These employees are all members of the Marketing database role. To support this application, you create a view named vwCustomers on the Customers table. After creating the view, you grant SELECT permissions on the view to the Marketing role.

When members of the Marketing role attempt to retrieve data from the view, they receive the following error message:

```
SELECT permission denied on object 'Customers', database 'Corporate', owner
'Marc'.
```

You must ensure that the members of the Marketing role can only use the vwCustomers view to access the data in the Customers table. What should you do?

A. Add the marketing role to the sysadmin fixed server role.

B. Transfer the ownership of the vwCustomers view to the marketing role.

C. Instruct Marc to transfer the ownership of the Customers table to each member of the marketing role.

D. Instruct Marc to grant the users SELECT permissions on the Customers table.

E. Drop the vwCustomers view. Instruct Marc to re-create the view and to grant SELECT permissions on the view to the marketing role.

Answer: E.
Explanation: This is an example of a broken permission chain. Marc is the owner of the table, but you are the owner of the view. As Marc is the database object owner of the table, he either has to grant other users permission to create views or has to create the view himself. As the table contains confidential information, granting others permission to create views would represent a potential breach in security since, in order to create views, the user would require explicit SELECT permission. Therefore Marc must create the view. However, to prevent possible database conflicts and possible compromises in data integrity, you must drop the view you created before Marc can re-create the view.

Incorrect answers:
A: Adding the Marketing role to the sysadmin role would give its members too much permissions and rights. They should only have SELECT permission on the vwCustomers view.
Note: Fixed server roles of which sysadmin is an example cannot be created and are defined at the server level. As such they exist outside of individual databases. A user with a SQL Server or Windows NT 4.0 or Windows 2000 login account can be added to the fixed server role and any member of a fixed server role can add other logins. Windows NT 4.0 and Windows 2000 users who are members of the BUILTIN\Administrators group are automatically added to the sysadmin fixed server role. Furthermore, members of the sysadmin role can perform any action on the server including CREATE VIEW, etc.

B: By transferring ownership of the vwCustomers view to the Marketing role, its members would be able to access the view as required. They would, however, also be able to alter the view, which could violate the security.

Note: A user who creates a database object such as a table, index, view, trigger, etc, is the database object owner for that database object. Permission to create database objects must be given by the database owner or system administrator. However, after these permissions are granted, a database object owner can create an object and grant other users permission to use that object.

C: Transferring ownership of the table to members of the Marketing role will grant those members explicit permissions on the table to create views and to grant permissions to others to use the table. This would represent a potential breach of security and would be of concern, as the table contains confidential information.

D: The grant object permissions statement allows a user account to perform activities or work with data in the current database and to restrict them from activities or information that is not part of their intended function. It is thus possible to grant SELECT object permission to certain columns on a table and not to others, thus restricting which columns may be used by the members of the marketing role. However, this solution applies the GRANT permission to the table. In this event, the SELECT permission is applied to all objects in the table. This would represent a potential breach of security and would be of concern, as the table contains confidential information.

Q. 17
You are designing your company's SQL Server 2000 sales database, which will be accessed by a custom application. Customer service and marketing employees require SELECT, INSERT, and UPDATE permissions on all tables in the Sales database. In addition to using the custom application, the marketing employees will use Microsoft Excel to retrieve data from the sales database to create charts.

Customer service employees belong to a Microsoft Windows 2000 group named CSR, and marketing employees belong to a Windows 2000 group named Marketing.

You want the customer service employees to access the sales database only from within the custom application. You want to allow the marketing employees to use both the custom application and Excel to access the sales database. No permissions have been granted in the database.

What should you do?

A. Create a database role named **Marketing** for the marketing employees and a database role named **CustomerService** for the customer service employees.
Add the Windows 2000 groups to the user-defined database roles.
Grant SELECT permissions on all tables in the database to the **Marketing** database role.
Grant SELECT, INSERT, and UPDATE permissions on all tables in the database to the **CustomerService** database role.

B. Create one application role to be used by both the customer service and marketing employees.
Grant SELECT permissions on all tables in the database to the Windows 2000 Marketing group.
Grant SELECT, INSERT, and UPDATE permissions on all tables in the database to the Windows 2000 CSR group.

C. Create an application role for the custom application.
Grant SELECT, INSERT and UPDATE permissions to the application role. Create a database role

named **Marketing** for the marketing employees.
Add the Windows 2000 Marketing group to the **Marketing** database role.
Grant SELECT permissions on all tables in the database to the **Marketing** database role.

D. Create one application role for the customer service employees.
Create a second application role for the marketing employees.
Grant SELECT permissions on all tables in the database to the application role to be used by the marketing employees.
Grant SELECT, INSERT, and UPDATE permissions on all tables in the database to the application role to be used by the customer service employees.

E. Create one application role for the customer service employees.
Create a second application role for the marketing employees to access the **Sales** database by using the custom application.
Create a third application role for the marketing employees to use when retrieving data from Excel.
Grant SELECT permissions on all tables in the database to both application roles to be used by the marketing employees.
Grant SELECT, INSERT, and UPDATE permissions on all tables in the database to the application role to be used by the customer service employees.

Answer: C.
Explanation: Creating an application role for the custom application and granting SELECT, INSERT and UPDATE permissions to the application role would grant users access to the database through the use of the custom application. This meets the requirements of the first task in this scenario.

By creating a database role named Marketing for the marketing employees, adding the Windows 2000 Marketing group to the Marketing database role, and granting SELECT permissions on all tables in the database to the Marketing database role, Excel could use this database role to access the database. Only the Marketing group would be allowed to use Excel like this and they would only have SELECT permissions.

Note: This scenario contains two tasks: the first is restricting customer service employees access to the sales database so that they gain access to the database only from within the custom application, and the second is to allow the marketing employees to use both the custom application and Excel. Both of these tasks can be accomplished through the use of application roles. Application roles contain no members, and Windows NT 4.0 or Windows 2000 groups, users, and roles cannot be added to application roles. Permissions of the application role are gained only when the application role is activated for the user's connection through a specific application or applications. In other words, a user's association with an application role is due to his or her ability to run the application. Furthermore, application roles are inactive by default and require a password to be activated, and they bypass standard permissions. Thus, when an application role is activated, i.e. when the user runs the application, all permissions applied to the user's login, user account, or as a member of a group or database role in all databases are temporarily suspended for the duration of the connection, and the permissions associated with the application role for the database in which the application role exists are brought to bear on the connection. This ensures that all the functions of the application can be performed regardless of the permissions that apply to the user. Users and Windows NT 4.0 or Windows 2000 groups do not need to be granted any permissions for the database because all permissions can be assigned by the applications they use to access the database. In such an event, a system-wide password can be assigned to an application role if access to the applications is secure.

Incorrect answers:

A: The use of database roles are inappropriate for this scenario, as permissions assigned to these roles would grant users direct access to the database without requiring them to make use of the application. This solution thus does not restrict the method through which the database may be accessed.

B: Creating one application role to be used by both the customer service and marketing employees would grant them access to the database through the use of the custom application if the custom application is assigned the application role. However, the customer service employees need INSERT and UPDATE permission, not only SELECT permission. Furthermore, the permission should be granted to the application role, not to the groups.

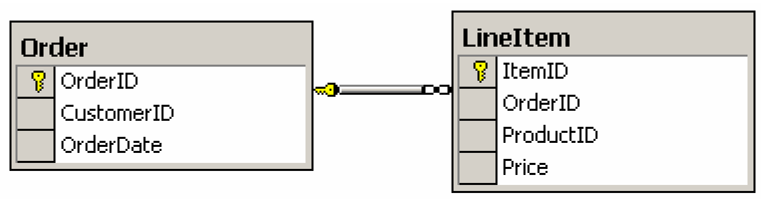
D: Creating one application role for the customer service employees would permit them to access the database only through the custom application, if the custom application were assigned this database role. Creating a second application role for the marketing employees would permit customer service employees to access the database only through either the custom application or Excel, depending on which application is assigned the application role. The marketing employees, however, will not have access to the database through the other application. Furthermore the marketing group would only have SELECT permission. They need INSERT and UPDATE permissions as well.

E: Through the custom application the marketing employees are required to have the SELECT, INSERT, and UPDATE permissions on all tables in the Sales database. However, by the proposed solution only SELECT permissions on all tables in the database are granted to both application roles to be used by the marketing employees.

Q. 18

You are a database developer for an online book retailer. Customers use the company's web site to place orders for books. As orders are entered, they are inserted into a database named BookOrders. During a nightly batch process, the order information is transferred to a database named Reports.

The Reports database includes a table named Order and a table named LineItem. The Order table contains basic information about the orders. The LineItem table contains information about the individual items in the orders. The Order and LineItem tables are shown in the exhibit.



Customers must be able to use the company's web site to view orders stored in the Reports database. Customers should be able to see only their own orders.

Customers should not be able to modify the orders. The primary key values of the orders are not relevant to the customers and should not be visible.

What should you do?

A. Create a view that displays the order information for a customer.

B. Create a stored procedure that retrieves the order information for a customer.

C. Create a scalar user-defined function that retrieves the order information for a customer.

D. Grant SELECT permissions on the **Order** and **LineItem** tables to the customers.

Answer: B.
Explanation: A stored procedure could be used to retrieve order information from the two tables, Order and LineItem. The procedure could look something like this:

```
CREATE PROCEDURE CustomerInfo (@CustID as Integer)
AS
SELECT OrderDate, ProductID, Price
FROM Order
INNER JOIN LineItem
ON Order.OrderID = LineItem.OrderID
WHERE Order.CustomerID=@CustID
```

This procedure would only show order information specific to a single customer. It would not give customers any possibility to update any information.

Note: A stored procedure is a group of Transact-SQL statements compiled into a single execution plan that can return data as OUTPUT parameters, which can return either data such as an integer or character value or a cursor variable; as return 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 and can also improve performance, as they contain many tasks that would otherwise be 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. Instead the application of the conditional logic and execution of the task is performed by the stored procedure and is done on the server. Because stored procedures supports all of the business functions that users may need to perform, users can execute the stored procedures that model the business processes with which they are familiar without having to access the table directly.

Incorrect answers:

A: We could create a view:

```
CREATE VIEW CustomerInfo
AS
SELECT OrderDate, ProductID, Price
FROM Order
INNER JOIN LineItem
ON Order.OrderID = LineItem.OrderID
WHERE Order.CustomerID=@CustID
WITH CHECK OPTION
```

We use the local variable @CustID to make the view only show information of a specific customer. The view could, however, allow the customer to update or delete information through the view. The customers are not allowed to modify the tables.

Note: A view is a virtual table and can be used to access data through the use of the SELECT statement. A user can use this virtual table by referencing the view name in Transact-SQL statements the same way a table is referenced. A view can be used to restrict a user to specific rows and/or columns in a table and to join columns from multiple tables so that they appear as a single table. While this would meet the requirement for a single customer, multiple views would have to be created for multiple customers. This would increase database maintenance overhead that could be reduced by the use of stored procedures, as executing a stored procedure is more efficient than executing multiple SQL statements. This is because SQL Server does not have to compile the execution plan completely, while a set of SQL statements has to execute completely. Furthermore, the fully compiled execution plan for the stored procedure is retained in the SQL Server procedure cache; hence subsequent executions of the stored procedure could use the cached execution plan.

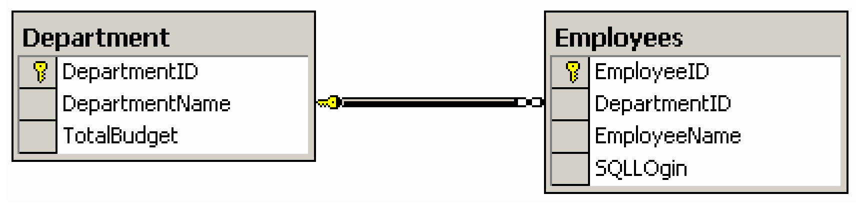
C: A scalar function can only produce scalar values. In this scenario we need a rowset result. We need the columns OrderDate, ProductID and Price.

Note: SQL Server 2000 supports three types of user-defined functions: scalar functions, inline table-valued functions and multi-statement table-valued functions. User-defined functions do not support output parameters. Scalar functions return a single data value of the type defined in a RETURNS clause. All scalar data types, including bigint and sql_variant, can be used. The timestamp data type, user-defined data type, and nonscalar types, such as table, view, or cursor, are not supported.

D: Granting SELECT permissions on the Order and LineItem tables to the customers would allow customers to SELECT all columns in these tables, thus allowing them to view their own orders and other customers' orders.

Q. 19

You are a database developer for a large travel company. Information about each of the company's departments is stored in a table named Department. Data about each of the company's travel agents and department managers is stored in a table named Employees. The SQLLogin column of the Employees table contains the database login for the travel agent or department manager. The Department and Employees table are shown in the exhibit.



Each department manager has been added to the Managers database role. You need to allow members of this database role to view all of the data in the department table. Members of this role should be able to insert or update only the row that pertains to their department.

You grant the Managers database role SELECT permissions on the Department table. What should you do next?

A. Create a trigger on the **Department** table that checks whether the database login of the user performing the insert or update operation belongs to a member of that department.

B. Create a view that includes all columns in the **Department** table and the **SQLLogin** column from the **Employees** table.

C. Include the WITH CHECK OPTION clause in the view definition.

D. Grant INSERT and UPDATE permissions on the **Department** table.

E. Grant INSERT and UPDATE permissions on the **SQLLogin** column of the **Employees** table.

Answer: B.
Explanation: All users of the Managers database role have SELECT permission on the department table. We must now allow members of this role to insert and update only rows with the appropriate department. We implement row-level security. This can be achieved by using a view that uses the WITH CHECK OPTION. The next step in the solution – we don't have to provide a complete solution, just the next step – would be to grant INSERT and UPDATE permissions to the view to the managers database role.

Note: The view could be implemented by**:**

```
CREATE VIEW Solution
AS
SELECT DepartmentID, DepartmentName, TotalBudget, SQLLogin
FROM Department INNER JOIN Employees
ON Department.DepartmentID = Emplyees.DepartmentID
WHERE SQLLogin = USER_NAME()
WITH CHECK OPTION
```

USER_NAME() finds the name of the current user.

By default, as rows are added or updated through a view, they disappear from the scope of the view when they no longer fall into the criteria of the query defining the view. The WITH CHECK OPTION clause forces all data modification statements that are executed against a view to adhere to the criteria set within the SELECT statement. When a row is modified through the view, the WITH CHECK OPTION ensures that the rows cannot be modified in a way that causes them to disappear from the view. Any modification that would cause this to happen is cancelled and an error message is displayed. By using the WITH CHECK OPTION and setting the view definition to specify that the managers' logins in the SQLLogin column are referenced when determining whether a particular user is allowed to alter the data in a particular row, managers will be prevented from modifying the data in rows that do not pertain to their departments.

Reference:
BOL**,** Create View
BOL; user_name

Incorrect answers:
A: It might be possible to create a trigger that implements row-level security; however, triggers should only be used as a last resource when no better solution is at hand. Views are the best method to implement row-level security.

C: Granting INSERT and UPDATE permissions on the department table will allow users to alter the data in any row or column on that table. Managers will thus be able to insert or update on any row in the table regardless of whether those rows pertain to the manager's department.

D: In this solution INSERT and UPDATE permissions are granted on the SQLLogin column only. Consequently users will only be able to INSERT or UPDATE data in that column. This solution thus does not allow managers to perform INSERT or UPDATE operations against the row that pertains to their department.

Q. 20
You are a database developer for your company's database named Sales. The database contains a table named Orders. The script that was used to create the table is shown in the Script Orders Table exhibit.

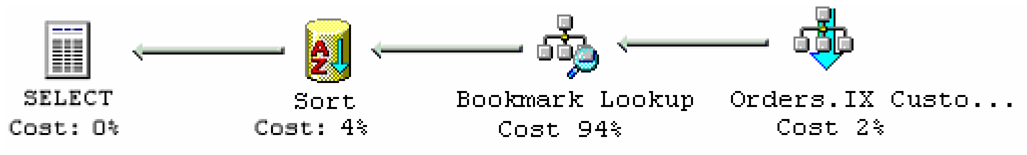
```
CREATE TABLE Orders
     (
     OrderID int NOT NULL,
     CustomerID char (5) NOT NULL,
     OrderDate datetime DEFAULT GETDATE ( ) NULL,
     ShippedDate datetime NULL,
     Freight money NULL,
     ShipName varchar (40) NULL
     )
GO
CREATE CLUSTERED INDEX IX_OrderID ON Orders (OrderID)
GO
CREATE NONCLUSTERED INDEX IX_CustomerID ON Orders (CustomerID)
```

An application will execute queries like the following to retrieve orders for a customer:

```
SELECT OrderID, CustomerID, OrderDate
FROM Orders WHERE CustomerID = 'WHITC'
ORDER BY OrderDate DESC
```

The query execution plan that is generated is shown in the Query Execution Plan exhibit. .

Query Execution Plan



You want this query to execute as quickly as possible. What should you do?

A. Create a nonclustered index on the OrderDate column.

B. Create a clustered index on the OrderDate column.

C. Change the clustered index on the OrderID column to a nonclustered index.

D. Change the nonclustered index on the CustomerID column to include the OrderDate column.

Answer: D.
Explanation:
The query is currently being run as follows:

1. An indexed scan on the CustomerID column.
2. A bookmark lookup on the OrderDate column.

The Bookmark Lookup is very costly (94%). We must index the OrderDate column. By changing the unclustered index on the CustomerID column to a composite unclustered index that includes both the CustomerID and the OrderDate columns, the query could execute in only one index scan. The WHERE clause and the ORDER clause could use the same index scan.

Note: The Bookmark Lookup logical and physical operator uses a bookmark (row ID or clustering key) to look up the corresponding row in the table or clustered index.

Incorrect answers:
A: By adding a nonclustered index to the OrderDate column, the Bookmark Lookup would be replaced by an index scan. This would improve performance. Though combining the OrderDate and the CustomerID column into a single composite index would be even more beneficial.

B: There is already a clustered index. There can be only one clustered index in a table.

C: The OrderID column is not used in the WHERE clause. The ordering is done on the OrderDate column.

Q. 21
You are database developer for your company's SQL Server 2000 database named Sales. The company has several custom web-based applications that retrieve data from the Sales database. Some of these applications use the EXECUTE statement to allow users to issue ad hoc queries. As the use of the Web-based applications increases, queries are taking longer to execute.

You want to discover which applications are sending a high number of these queries to the database server. What should you do?

A. Use SQL profiler to capture the **RPC:Completed** event.
Group the trace by the **HostName** data column.

B. Use SQL profiler to capture the **SQL:StmtCompleted** event.
Group the trace by the **ApplicationName** data column.

C. Use System Monitor to monitor the **SQLServer:Database** counter.
Select all counters for the **Sales** database

D. Use System Monitor to monitor the **SQLServer:General Statistics** counter.
Select all counters for the **Sales** database.

Answer: B.
Explanation: SQL Profiler is a graphical tool that allows system administrators to monitor events in an instance of SQL Server. Data pertaining to an event can be captured and saved to a file or SQL Server table to be analyzed at a later date. SQL Profiler can filter events to monitor SQL statements that have completed by using the **SQL:StmtCompleted** event and specifying which columns to use in the trace. In this solution, specifying that SQL profiler should use the ApplicationName data column with the **SQL:StmtCompleted** event will indicate how many queries that a particular application has sent to an instance of SQL have been completed, thus indicating how many queries a particular application has run.

Reference:
INF: Troubleshooting Performance of Ad-Hoc Queries (Q243588)
INF: Troubleshooting Application Performance with SQL Server (Q224587)

Incorrect answers:
A: SQL Profiler can filter events to monitor remote stored procedures (RPC) that have completed by using **RPC:Completed** event and specifying which columns to use in the trace. In this solution, specifying that SQL profiler should use the HostName data column with the **RCP:Completed** event will indicate how many queries that originate from a particular host have been completed. This will thus indicate how many queries a particular host has run on an instance of SQL Server.

C: System Monitor is used to generate statistics and alerts. System Monitor cannot be used to monitor SQL queries from a specific application.

Note: SQL Server provides objects and counters that can be used by System Monitor in Windows 2000 or by Performance Monitor in Windows NT 4.0 to monitor activity on computers running an instance of SQL Server. The **SQLServer:Database** object provides counters to monitor bulk copy operations, backup and restore throughput, and transaction log activities. Transactions can be monitored and the transaction log can be used to determine how much user activity is occurring in the database and how full the transaction log is becoming. The amount of user activity can determine the performance of the database and affect log size, locking, and replication. Monitoring low-level log activity to gauge user activity and resource usage can help identify performance bottlenecks.

D: System Monitor is used to generate statistics and alerts. System monitor cannot be used to monitor SQL queries from a specific application.

Note: SQL Server provides objects and counters that can be used by System Monitor in a Windows 2000 or by Performance Monitor in Windows NT 4.0 to monitor activity on computers running an instance of SQL Server. The **SQLServer:General Statistics** object provides counters to monitor general server-wide activity, such as the number of current connections and the number of users connecting and disconnecting per second from computers running an instance of SQL Server. This can be useful when working on large online transaction processing (OLTP) type systems where there are many clients connecting and disconnecting from an instance of SQL Server.

Q. 22
You are a database developer for a multinational corporation. The company has a centralized online transaction processing database located on a SQL Server 2000 computer. This database has a table named Sales, which contains consolidated sales information from the company's offices.

During the last year, more than 150,000 rows have been added to the Sales table. Users of the database report that performance during the course of the year has steadily decreased.

You need to improve the performance of queries against the Sales table. In the SQL query analyzer, which script should you execute?

A. EXEC sp_updatestats 'resample'

B. CREATE STATISTICS Sales WITH FULLSCAN

C. Sp_autostats 'Sales'

D. UPDATE STATISTICS Sales WITH FULLSCAN ALL

Answer: A.
Explanation: 150,000 rows have been added. We should make sure that the statistics are up to date. We could either create statistics or update them.

The sp_updatestats command runs UPDATE STATISTICS against all user-defined tables in the current database. It may seem unnecessary to update the statistics on all user tables, but the other alternatives are incorrect.

The 'resample' argument specifies that sp_updatestats will use the RESAMPLE option of the UPDATE STATISTICS command. New statistics will inherit the sampling ratio from the old statistics.

The `EXEC sp_updatestats 'resample'` code is correct and does not contain any syntactical errors. It would run and update statistics on all user tables in the database including the Sales table.

Note: SQL Server keeps statistics about the distribution of the key values in each index and uses these statistics to determine which index(es) to use in query processing. Query optimization depends on the accuracy of the distribution steps.

Incorrect answers:

B: The CREATE STATISTICS command creates a histogram and associated density groups (collections) over the supplied column or set of columns. The proposed solution includes a syntax error, however. We must specify a statistics name, and on which columns the statistics should be computed. The following code is valid:

```
CREATE STATISTICS statistics_name ON Sales (SalesID) WITH FULLSCAN
```

.

C: sp_autostats displays or changes the automatic UPDATE STATISTICS setting for a specific index and statistics, or for all indexes and statistics for a given table or indexed view in the current database to either off or on. It does not UPDATE STATISTICS by itself but can turn on the automatic UPDATE STATISTICS procedure. This update is performed by the task that detected that the statistics needed to be updated. The update is performed using a complex sampling method that minimizes the effect of the update on transaction throughput.

D: The syntax of the statement is incorrect; there is a missing comma (,). The correct statement would be:

```
UPDATE STATISTICS Sales WITH FULLSCAN, ALL
```

Q. 23

You are a database developer for your company's SQL Server 2000 online transaction processing database. You have written several stored procedures that will produce critical sales reports. The stored procedures access existing tables, which are indexed.

Before you put the stored procedures in the production environment, you want to ensure optimal performance of the new stored procedures. You also want to ensure that daily operations in the database are not adversely affected.

What should you do?

A. Create a covering index for each query contained in the stored procedures.

B. For each query in the stored procedures, create an index that includes each column contained in the WHERE clause.

C. Use output from the Index Tuning Wizard to identify whether indexes should be added.

D. CREATE STATISTICS on all columns in the SELECT and WHERE clauses of each query.

Answer: C.
Explanation:
In this scenario we want to accomplish two goals:

1. You want to ensure optimal performance of the new stored procedures.
2. You also want to ensure that daily operations in the database are not adversely affected.

These two goals might be conflicting. Optimizing for only performance of the new stored procedure would increase the number of indexes, which would slow down updates and insertions of rows in the OLTP database.

The Index Tuning Wizard can take these conflicting goals into account and produce a tradeoff.

Note: SQL Profiler and the Index Tuning Wizard should be used to help analyze queries and determine which indexes should be created on a table. 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. The Index Tuning Wizard can be used to automate this task.

Incorrect answers:
A: Creating covered indexes for the stored procedures would improve performance of these procedures but it would also slow down updates and insertions of rows in the OLTP database.

Note: Covered queries are queries where all the columns specified in the query are contained within the same index. Covered queries can greatly improve performance because all the data for the query is contained within the index itself. Thus, only the index pages and not the data pages of the table must be referenced to retrieve the data. This reduces overall I/O. Although adding columns to an index to cover queries can improve performance, maintaining the extra columns in the index incurs update and storage costs.

B: As a general rule, creating indexes for columns that appear in a WHERE clause improves performance. In this dynamic scenario it would be better to let the Index Tuning Wizard decide which indexes are beneficial since too many indexes would slow down an OLTP database.

D: Only the table owner can CREATE STATISTICS on a table. The owner of a table can create a statistics group at any time, whether there is data in the table or not. CREATE STATISTICS can be executed on views with clustered index, or indexed views. Statistics on indexed views are used by the optimizer only if the view is directly referenced in the query and the NOEXPAND hint is specified for the view. Otherwise, the statistics are derived from the underlying tables before the indexed view is substituted into the query plan. Such substitution is supported only on Microsoft SQL Server 2000 Enterprise and Developer Editions.

Q. 24
You are a database developer for an insurance company. You are informed that database operations, such as selects, inserts, and updates, are taking much longer than they were when the database was created a year ago.

The previous developer added necessary indexes on the tables when the database was created. Since that time, additional tables and stored procedures have been added to the database. In addition, many of the queries are no longer used.

You want to improve the response time of the database operations as quickly as possible. What should you do?

A. Execute the DBCC UPDATEUSAGE statement against the database to update the sysindexes system table.

B. Execute the DBCC SHOW_STATISTICS statement to find high-density indexes. Drop the high-density indexes.

C. Run the Index Tuning Wizard against a workload file to suggest indexes to create and drop the suggested indexes.

D. Use SQL profiler to find table scans. Add indexes to tables that were found to have table scans.

Answer: C.
Explanation: The Index Tuning Wizard can be used to analyze queries and determine which indexes should be created on a table and 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. 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. The Index Tuning Wizard can be used to automate this task. 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 a 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.

Incorrect answers:

A: Correcting inaccuracies in the sysindex tables with the DBCC UPDATEUSAGE command would not improve query performance. It would only help correcting the output of the sp_spaceused stored procedure

Note: DBCC UPDATEUSAGE reports and corrects inaccuracies in the sysindexes table, which may result in incorrect space usage reports by the sp_spaceused system stored procedure. It corrects the rows, used, reserved, and dpages columns of the sysindexes table for tables and clustered indexes. Size information is not maintained for nonclustered indexes. If there are no inaccuracies in sysindexes, DBCC UPDATEUSAGE returns no data. UPDATEUSAGE can be used to synchronize space usage counters. DBCC UPDATEUSAGE can take some time to run on large tables or databases and thus should be used only when it is suspected that incorrect values are being returned by sp_spaceused.

B: Dropping high density indexes would most likely decrease the performance. Useful indexes improve performance.

Note: All indexes in a table have distribution statistics that describe the selectivity and distribution of the key values in the index. Selectivity is a property that relates to how many rows are usually identified by a key value. A unique key has high selectivity; a key value found in numerous rows has poor selectivity. The selectivity and distribution statistics are used by SQL Server 2000 to optimize its navigation through tables and indexed views when processing Transact-SQL statements. Furthermore, the distribution statistics are used to estimate how efficient an index would be in retrieving data associated with a key value or range specified in the query. A fill factor is a property of a SQL Server index that controls how densely the index is packed and is only applied when the index is created. The default fill factor normally delivers good performance, but in some cases it may be beneficial to change the fill factor. If the table is envisaged to have many updates and inserts, an index with a low fill factor should be created in order to leave more room for future keys. If the table will be a read-only table that will not change, creating an index with a high fill factor to reduce the physical size of the index, which will in turn reduce the number of disk reads SQL Server need to navigate through the index. As keys are inserted and deleted, the index will eventually stabilize at a certain density.

D: SQL Profiler can be used to monitor when a table or index is being scanned during the execution of a query. By using the **Scan:Started** and **Scan:Stopped** event classes, it is also possible to monitor the type of scans being performed by a query. However, this solution only suggests adding indexes. While this may improve SELECT response time for tables that were found to have table scans, it does not have any effect on other tables. Furthermore, indexes take up disk space and can slow the adding, deleting, and updating of rows. Thus, any indexes that infrequently used by queries should be dropped to reduce maintenance overhead.

Q. 25
You are a database developer for a sporting goods company. The company has one main office and many regional offices across the United States. A 56-Kbps frame relay network connects the offices. Each office has a SQL Server 2000 database that contains information about the company's products. The main office SQL Server databases is used to process incremental updates to the regional office databases. Transactional replication is used to perform these updates.

Each quarter, you create a new snapshot file. You use this snapshot file to replace the regional office databases with the latest product information. This snapshot file is now more than 800 MB in size. You need to apply the snapshot file to the regional office databases. You want to accomplish this by using the least amount of expense.

What should you do?

A. Use transactional replication.
Mark all subscriptions for reinitialization.

B. Copy the snapshot to an NTFS compressed share.
Create a CD from that share.
Distribute the CD to the regional office.

C. Create a compressed snapshot in the default snapshot folder on the Distributor.
Create CD from that folder.
Distribute the CD to the regional offices.

D. Create a compressed snapshot in an alternative snapshot folder on the network.
Create CD from that folder.
Distribute the CD to the regional offices.

E. Create a compressed snapshot in an alternative snapshot folder on the network.
Use FTP to distribute the snapshot to the regional offices.

Answer: D.
Explanation: It is possible to compress a snapshot file in the Microsoft CAB file format. When the snapshot file is too large to fit on removable media, it must be transmitted over a slow network. Compressing a snapshot file can reduce network traffic but will increases the time required to generate and apply the snapshot. However, snapshot files can only be compressed when they are saved to an alternate location or when subscribers are accessing them through FTP. Snapshot files written to the default snapshot folder on the distributor cannot be compressed.

In this scenario we make the optimal solution by choosing to distribute the compressed snapshot file on CD's.

Incorrect answers:

A: With transactional replication, an initial snapshot of data is applied to subscribers, and then when data modifications are made at the publisher, the individual transactions are captured and propagated to subscribers. Transactional replication is helpful when incremental changes must be propagated to subscribers as they occur, when transactions need to adhere to ACID properties, and when subscribers are reliably and/or frequently connected to the publisher. If subscribers need to receive data changes in near real-time, they need to be connected to the Publisher. In this scenario, the subscribers, i.e. the regional offices, are connected to the publisher, i.e. the main office, via a slow 56-Kbps frame relay network.

B: NTFS compressed folders cannot be copied to non-NTFS file systems such as FAT or FAT32 that are used by Windows 9x, or CDFS that are used by CDs without the file losing its compression status, i.e. uncompressing. Thus, the file will be too large to copy to the CD.

C: It is possible to compress a snapshot file in the Microsoft CAB file format. When the snapshot file is too large to fit on removable media, it must be transmitted over a slow network. Compressing a snapshot file can reduce network traffic but will increases the time required to generate and apply the snapshot. However, snapshot files can only be compressed when they are saved to an alternate location or when subscribers are accessing them through FTP. Snapshot files written to the default snapshot folder on the distributor cannot be compressed.

E: It would be cheaper to distribute the snapshot on CDs compared to FTP on slow 56-Kbps WAN links. Distributing through Internet connection would cost bandwidth and money.

Q. 26

You are a database developer for your company's SQL Server 2000 database. This database contains a table named Sales, which has 2 million rows. The Sales table contains sales information for all departments in the company. Each department is identified in the table by the DepartmentID column. Most queries against the table are used to find sales for a single department.

You want to increase the I/O performance of these queries. However, you do not want to affect the applications that access the table.

What should you do?

A. Create a new table, and move the columns that are most frequently queried to this table.
Retain the **DepartmentID** column in both tables.
Create a view on the original table and on the new table.
Add a FOREIGN KEY constraint on the join columns of the new table.

B. Create a new table, and move the columns that are most frequently queried to this table.
Retain the **DepartmentID** column in both tables.

Create a view on the original table and on the new table.
Add a CHECK constraint on the **DepartmentID** columns of both tables.

C. Create one new table for each department, and move the sales information for each department to that department's table.
Add a CHECK constraint on the **DepartmentID** columns of the new tables.
Create a view on the new tables.

D. Create one new table for each department, and move the sales information for each department to that department's table.
Create a view on the new tables.
Add a CHECK constraint on the **DepartmentID** column in the view.

E. Create a stored procedure that accepts an input parameter for the department.
Use the stored procedure to return results from the Sales table.

Answer: C.
Explanation: To improve performance, the huge Sales table will be split into several tables. A horizontal split would be the most beneficial since most queries run are used to find sales for a particular department. In order to keep existing applications functioning, a view will be used which mimics the behavior of the original unsplit Sales table.

Deleting the original Sales table and naming the view Sales would allow applications to use sales data in the same way as before.

Incorrect answers:
A: The original table is not needed; in fact, keeping the original table would store the same information in two places, which always is a bad idea in database design.

B: Vertical splitting (column splitting) is not as good as horizontal splitting (row splitting) in this scenario. We have no indication that some columns are used less frequently.

D: You can't put a check constraint on a column in a view, only to a column in a table.

E: To increase performance, the sales tables table must be split. Just using a stored procedure on the existing Sales table will not improve I/O performance.

Q. 27
You are a database developer for your company's SQL Server 2000 database. You are deleting objects in the database that are no longer used. You are unable to drop the 1997Sales view. After investigation, you find that the view has the following characteristics:

- ∉ **There is a clustered index on the view**
- ∉ **The sales database role has permissions on the view.**
- ∉ **The view uses the WITH SCHEMABINDING option.**
- ∉ **A schema-bound inline function references the view**
- ∉ **An INSTEAD OF trigger is defined on the view**

What should you do before you can drop the view?

A. Drop the clustered index on the view.

B. Remove all permissions from the view.

C. Remove the WITH SCHEMABINDING option from the view.

D. Remove the WITH SCHEMABINDING option from the function that is referencing the view.

E. Disable the INSTEAD OF trigger on the view.

Answer: D.
Explanation: The CREATE FUNCTION supports a SCHEMABINDING clause that binds a function to the schema of an object, such as a table, a view, or another user-defined function that it references. An attempt to alter or drop any object referenced by a schema-bound function will fail. The ALTER FUNCTION can be used to remove the SCHEMABINDING clause by redefining the function without specifying the WITH SCHEMABINDING clauses.

Incorrect answers:
A: A clustered index on the view would not prevent it from being deleted.

Note: A clustered index alters the physical storage order of the data in the table. A table can contain only one clustered index and can comprise multiple columns. A clustered index is particularly efficient on columns that are often searched for a range of values. After the row with the first value is found using the clustered index, rows with subsequent indexed values are guaranteed to be physically adjacent. If a column is frequently used to sort the data retrieved from a table, it can be advantageous to cluster the table on that column to save the cost of a sort each time the column is queried. When an index is no longer needed, it can be deleted from a database and the storage space it uses can be reclaimed. Only the owner of a table can delete indexes on that table, and the owner of the table cannot transfer this permission to other users. However, members of the db_owner and db_ddladmin fixed database roles and sysadmin fixed server role can drop any database object by explicitly specifying the owner of the object in the DROP VIEW statement. Furthermore, indexes created on any views or tables are automatically deleted when the view or table is dropped, and since the DROP VIEW statement can be executed against indexed views, indexes do not prevent the dropping of a view.

B: A view can be deleted if it is no longer needed, or if the view definition and the permissions associated with it need to be cleared. When a view is deleted, the tables and the data upon which it is based are not affected. Thus, the permissions that pertain to a view do not prevent the view from being dropped.

C: The SCHEMABINDING clause is supported by the CREATE VIEW statement and binds the view to a schema. Views or tables participating in a view created with the schema binding clause cannot be dropped unless that view is dropped or changed so that it no longer has schema binding. Dropping the SCHEMABINDING on the view would make it possible to drop those other views or tables. This SCHEMABINDING does not, however, prevent the view itself from being dropped.

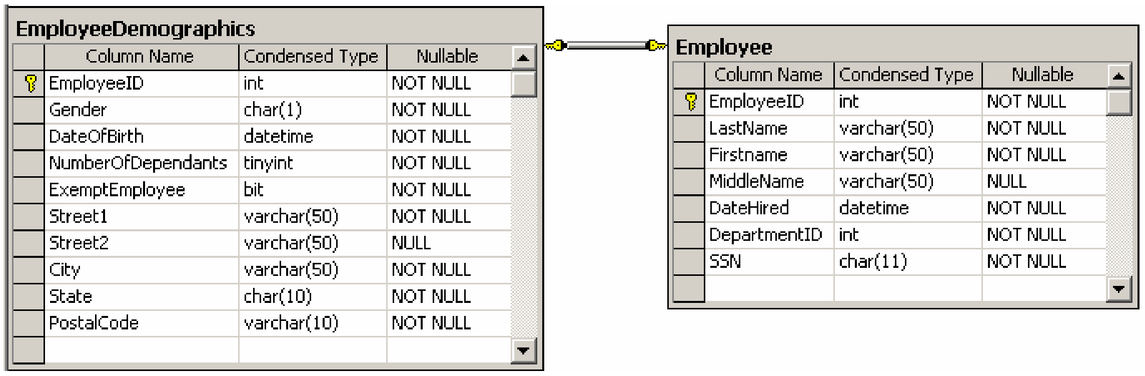
E: Disabling a trigger on the view would not affect the possibility of dropping the view.

Note: Triggers are a special type of stored procedure that execute automatically when an UPDATE, INSERT, or DELETE statement is issued against a table or view. INSTEAD OF triggers allow views that are not updateable to support updates. A view comprising multiple base tables must use an INSTEAD OF trigger to support inserts, updates and deletes that reference data in the tables. A trigger can be deleted when it is no longer needed. When a trigger is deleted, the table and the data upon which it is based are not affected. While permissions to delete a trigger default to the owner of the table or view upon which the trigger is defined, deleting the table or view will automatically delete any triggers on that table or view.

Q. 28
You are a database developer for Proseware, Inc. You are creating a database named HumanResources for the company. This database will contain all employee records and demographics information. The company has 2,000 employees and experiences a yearly turnover rate of about 2 percent. When employees leave the company, all of their records must be retained for auditing purposes. Employee demographics information changes at a yearly rate of about 9 percent. You do not need to maintain a history of demographics changes.

The schema for the human resources database is shown in the human resources schema exhibit, and the scripts that will be used to create the indexes are shown in the Index Scripts exhibit.

HumanResources Schema:



EmployeeDemographics

Column Name	Condensed Type	Nullable
EmployeeID	int	NOT NULL
Gender	char(1)	NOT NULL
DateOfBirth	datetime	NOT NULL
NumberOfDependants	tinyint	NOT NULL
ExemptEmployee	bit	NOT NULL
Street1	varchar(50)	NOT NULL
Street2	varchar(50)	NULL
City	varchar(50)	NOT NULL
State	char(10)	NOT NULL
PostalCode	varchar(10)	NOT NULL

Employee

Column Name	Condensed Type	Nullable
EmployeeID	int	NOT NULL
LastName	varchar(50)	NOT NULL
Firstname	varchar(50)	NOT NULL
MiddleName	varchar(50)	NULL
DateHired	datetime	NOT NULL
DepartmentID	int	NOT NULL
SSN	char(11)	NOT NULL

```
ALTER TABLE [dbo].[Employee] WITH NOCHECK ADD
     CONSTRAINT [pk_Employee] PRIMARY KEY CLUSTERED
     ([EmployeeID])
     WITH FILLFACTOR = 90
GO
ALTER TABLE [dbo].[EmployeeDemographics] WITH NOCHECK ADD
     CONSTRAINT [dbo].[EmployeeDemographics] PRIMARY KEY CLUSTERED
     ([EmployeeID])
     WITH FILLFACTOR = 90
GO
```

You want to conserve disk space and minimize the number of times that expansion of the database files needs to occur. All varchar columns are 50 percent full.

Which two parameters should you specify for the CREATE DATABASE statement? (Each correct answer presents part of the solution. Choose two)

A. SIZE = 1GB

B. SIZE = 1MB

C. SIZE= 2048KB

D. FILEGROWTH = 20

E. FILEGROWTH = 5%

F. FILEGROWTH = 0

Answer: C, E.
Explanation:
C: We must decide the initial size and the file growth of the database. We calculate the specifics for this database and then add the size of the model database. All new databases are based on the model database. The model database has an initial size of 1 MB. A quick estimate of the user data of this table (see Note below) is that less than 1 MB is needed, so a total size of 2 MB is sufficient.

E: As the company experiences a yearly turnover of about 2% of its employee size, and because data on employees who have left the company must be maintained for auditing purposes, a FILEGROWTH of 5% should be an adequate rate of growth of the database.

Note: Estimate size of user data:

EmployeeDemographics table
We estimate the size of one row:
Varchar columns: total of 160. They are less than 50% full, so about 80 bytes would be needed.
Rest of columns. An estimate. Less than 30 bytes.
We conclude: one row = 110 bytes.

Employee table
Varchar columns: total of 150. They are less than 50% full. So about 75 bytes would be needed.
Rest of columns: An estimate. Around 30 bytes.
We conclude: one row = 105 bytes.

Both tables.
There is a one-to-one relationship between the tables, so we can simply add the estimates of the tables.
Both tables: 225 bytes
All rows (2000 rows): 2000*125 bytes = 250,000 bytes
Data pages only fill to 90%, due to fill factor. So we actually need approx. 278, 000 bytes (250,000/0.9).
If we add the size of the model database (1 MB) we see that 2 MB (or 2048 KB) would be more than enough and a good value for the default file size of the database.

Incorrect answers:
A: 1 GB would be too much. Only 2 MB are needed.

B: The model database has a size of at least 1 MB. A default file size of 1 MB would be too small.

D: FILEGROWTH specifies the growth increment of the file size and cannot exceed the MAXSIZE setting. If the FILEGROWTH value is not represented as a percentage, then the value represents a number of MB. A FILEGROWTH of 20 thus represents a FILEGROWTH of 20 MB.

F: FILEGROWTH specifies the growth increment of the file size and cannot exceed the MAXSIZE setting. If the FILEGROWTH value is not represented as a percentage, then the value represents a number of MB. SQL Server does not increment a file in units smaller than 1 MB. Furthermore, a SQL Server does not permit a FILEGROWTH of 0% or 0 MB.

Q. 29

You are a database developer for your company's SQL Server 2000 database. This database contains a table named Products and a table named Companies. You want to insert new product information from a linked server into the Products table. The Products table has a FOREIGN KEY constraint that references the Companies table. An UPDATE trigger is defined on the Products table.

You want to load the data as quickly as possible. What should you do?

A. Use the ALTER TABLE statement and the ON UPDATE clause to modify the **Products** table.

B. Use the ALTER TABLE statement and the DISABLE TRIGGER ALL option to modify the **Products** table.

C. Use the ALTER TABLE statement and the DISABLE TRIGGER ALL option to modify the **Companies** table.

D. Use the ALTER TABLE statement and the NOCHECK CONSTRAINT option to modify the **Companies** table.

E. Use the ALTER TABLE statement and the NOCHECK CONSTRAINT option to modify the **Products** table.

Answer: E.
Explanation: New products from a linked server must be inserted as quickly as possible on your company's database. The products should be inserted into the Products table, however there is a foreign key constraint from the Products table to the Companies table. This foreign key constraint will force overhead during the insertion of the new rows. By modifying the Products table by using the ALTER TABLE statement and the NOCHECK CONSTRAINT option, the data would be loaded much more quickly.

The idea is:
1. Remove the check constraints
2. Load the data
3. Add the CHECK constraints again.

Note 1: A drawback of disabling the foreign key constraint is that product rows with no corresponding company row in the Companies table could be entered. However it seems safe to assume that the new products all belong to already existing companies.

Note 2: The ALTER TABLE statement is used to modify a table definition by altering, adding, or dropping columns and constraints, or by disabling or enabling constraints and triggers. ALTER TABLE acquires a schema modify lock on the table to ensure no other connections reference the table during the modification. The modifications made to the table are logged and are fully recoverable. Modifications that affect all the rows in very large tables, such as dropping a column or adding a NOT NULL column with a default, can take a long time to complete and generate many log records.

The WITH NOCHECK clause specifies that the data in the table being updated must not be validated against a newly added or re-enabled FOREIGN KEY or CHECK constraint. If not specified, WITH CHECK is assumed for new constraints, and WITH NOCHECK is assumed for re-enabled constraints. The WITH CHECK and WITH NOCHECK clauses cannot be used for PRIMARY KEY and UNIQUE constraints.

The WITH NOCHECK clause can be used if you do not want to verify new CHECK or FOREIGN KEY constraints against existing data. This, however, is not recommended as any constraint violations suppressed by the WITH NOCHECK clause when the constraint is added may cause future updates to fail if they update rows that contain data that does not comply with the constraint but had been added while CHECK constraints had been suppressed by the WITH NOCHECK clause.

Incorrect answers:

A: If a row on the table that is being altered is updated through the ALTER TABLE statement has a referential relationship with a row on another table, the ON UPDATE clause can be used specify what action should occur to a row on the other table. The default is NO ACTION. This will raise an error when the row in the referential relation is updated through the ALTER TABLE statement, and the update action on the row in the table that is being altered will be rolled back. If CASCADE is specified, the row in the other table that has a referential relation with a row that is updated through the ALTER TABLE statement. The CASCADE action ON UPDATE cannot be defined if an INSTEAD OF trigger ON UPDATE already exists on the referenced table. As this solution does not specify what action to take, the update will revert to the default NO ACTION, and as the Products table has a FOREIGN KEY constraint that references the Companies table, this will raise an error, and the update action performed by the ALTER TABLE statement on the row on the Products the references a row on Companies table will be rolled back.

B: Disabling the triggers on the products could improve the performance somewhat, but there is only an UPDATE trigger on the Products and we are going to insert new data. The update trigger would not be used anyway, so disabling triggers would do no good.

C: The DISABLE TRIGGER ALL clause specifies that all triggers on a table being altered must be disabled. When a trigger is disabled it is still defined for the table; however, when INSERT, UPDATE, or DELETE statements are executed against the table, the actions in the trigger are not performed until the trigger is re-enabled. Furthermore, as it is the Products table that data is being inserted into, the triggers on the Products table and not the triggers on the Companies table should be disabled.

D: As data is to be added to the Products table and not the Companies table, the ALTER TABLE statement will not be run against the Companies table.

The WITH NOCHECK clause specifies that the data in the table being updated must not be validated against a newly added or re-enabled FOREIGN KEY or CHECK constraint. If not specified, WITH CHECK is assumed for new constraints, and WITH NOCHECK is assumed for re-enabled constraints. The WITH CHECK and WITH NOCHECK clauses cannot be used for PRIMARY KEY and UNIQUE constraints.

The WITH NOCHECK clause can be used you do not want to verify new CHECK or FOREIGN KEY constraints against existing data.

Q. 30
You are the database developer for your company's Accounting database. The database contains a table named Employees. Tom is a member of the accounting department. Tom's database user account has been denied SELECT permissions on the Salary and BonusPercentage columns of the Employees table. Tom has been granted SELECT permissions on all other columns in the table. Tom now requires access to all the data in the Employees table.

What should you do?

A. Revoke SELECT permissions on the **Salary** and **BonusPercentage** columns of the **Employees** table for Tom's database user account.

B. Add Tom to the **db_datareader** database role.

C. Add Tom to the **db_accessadmin** database role.

D. Grant SELECT permissions on the **Salary** and **BonusPercentage** columns of the **Employees** table for Tom's database user account.

Answer: D.
Explanation: Tom has the SELECT permission on all columns in the Employees table, except on the Salary and BonusPercentage columns. By simply granting Tom SELECT permission on those two columns, Tom would be able to us all columns of the Employees table.

Note: A granted permission removes the denied or revoked permission at the level granted (user, group, or role). The same permission denied at another level such as group or role containing the user takes precedence. However, although the same permission revoked at another level still applies, it does not prevent the user from accessing the object.

Incorrect answers:
A: Revoking the SELECT permissions on the Salary and BonusPercentage columns of the employees table would leave Tom with no permission of these two columns.

Note: this choice would work if Tom, as a member of a different group, had permissions on those two columns. But there is no indication of that. We cannot assume that such permission exists.

B: Adding Tom to the db_datareader database role will allow Tom to SELECT all the data from any user table in the database. This would give Tom more authority than is required, as Tom only requires access to all the data in the Employees table.

C: Adding Tom to the db_accessadmin database role will allow Tom to add user IDs or remove them from the database. This would give Tom more authority than is required, as Tom only requires access to all the data in the Employees table.

Q. 31
You are a database developer for a toy manufacturer. Each employee of the company is assigned to either an executive, administrative, or labor position. The home page of the company intranet displays company news that is customized for each position type. When an employee logs on to the company intranet, the home page identifies the employee's position type and displays the appropriate company news.

Company news is stored in a table named News, which is located in the corporate database. The script that was used to create the News table is shown in the exhibit.

```
CREATE TABLE News
          (
          NewsID int NOT NULL,
          NewsText varchar (8000) NOT NULL,
          EmployeePositionType char (15) NOT NULL,
          DisplayUntil datetime NOT NULL,
          DateAdded datetime NOT NULL DEFAULT (getdate( )),
          CONSTRAINT PK_News PRIMARY KEY (NewsID)
          )
```

Users of the intranet need to view data in the News table, but do not need to insert, update, or delete data in the table. You need to deliver only the appropriate data to the intranet, based on the employee's position type.

What should you do?

A. Create a view that is defined to return the rows that apply to a specified position type.

B. Create a stored procedure that returns the rows that apply to a specified position type.

C. Grant SELECT permissions on the **EmployeePositionType** column for each position type.

D. Grant permission on the **News** table for each position type.

Answer: B.
Explanation: We create a stored procedure with a parameter. The procedure could look like:

```
CREATE PROCEDURE NewsEmpType (@EmpType char(15))
AS
SELECT NewsText
FROM News
WHERE EmployeePositionType=@EmpType
```

The parameter is used to return only the rows for a specific position type. This procedure would produce the required rows. The employees would only be able to use it to return the rows, not to update or delete any rows.

By granting the employees the EXECUTE permission to the procedure they will be able to use it.

Note: A stored procedure is a group of Transact-SQL statements compiled into a single execution plan that returns data in four ways: as OUTPUT parameters, which can return either data such as an integer or character value or a cursor variable; as return 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 and can also improve performance, as they contain many tasks that would otherwise be 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 application of the conditional logic and execution of the task is performed by the stored procedure and is done on the server, thus improving efficiency and reducing database overhead.

Incorrect answers:

A: A view can be used for security, but is not as secure as a procedure. Users could use the view to update or delete rows in the table.

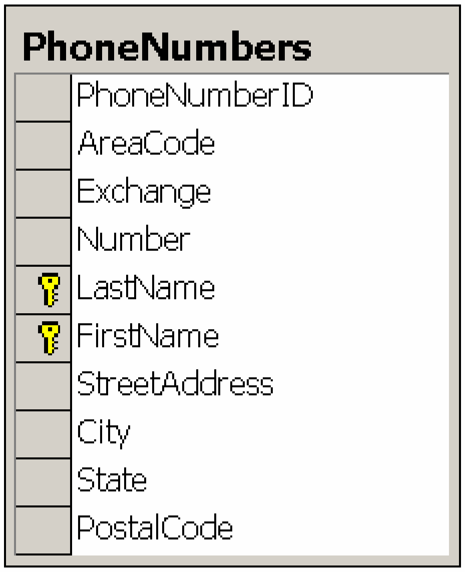
Note: A view is a virtual table and can be used to access data through the use of the SELECT statement. A user can use this virtual table by referencing the view name in Transact-SQL statements the same way a table is referenced. A view can be used to restrict a user to specific rows and/or columns in a table and to join columns from multiple tables so that they appear as a single table. While this would meet the requirement of the scenario, it is not the best solution.

C: Granting SELECT permissions on the EmployeePositionType column for each position type would only allow each position type to view the data in the EmployeePositionType column that applies to them, if specific roles are created on the basis of they various employee position types. This however will not allow them to SELECT the data that is not in those columns. They thus will not be able to view the news that pertains to their position types.

D: This solution is ambiguous. It does not state which permissions are granted. It however does not meet the requirements of the scenario, as the permissions would grant access to the entire table. Thus all the data, and not only those pertaining to the respective employee position types, could be accessed.

Q. 32

You are a database developer for a company that produces an online telephone directory. A table named PhoneNumbers is shown in the exhibit.



After loading 100,000 names into the table, you create indexes by using the following script:

```
ALTER TABLE [dbo]. [PhoneNumbers] WITH NOCHECK ADD
         CONSTRAINT[PK_PhoneNumbers]PRIMARY KEY CLUSTERED (
         [FirstName],
         [LastName],
) ON [PRIMARY]
GO
CREATE UNIQUE INDEX
     [IX_PhoneNumbers] ON [dbo].[PhoneNumbers](
     [PhoneNumberID]
) ON [PRIMARY]
GO
```

You are testing the performance of the database. You notice that queries such as the following take a long time to execute:

Return all names and phone numbers for persons who live in a certain city and whose last name begins with 'W'

How should you improve the processing performance of these types of queries? (Each correct answer presents part of the solution. Choose two.)

A. Change the PRIMARY KEY constraint to use the **LastName** column followed by the **FirstName** column.

B. Add a nonclustered index on the **City** column.

C. Add a nonclustered index on the **AreaCode**, **Exchange**, and **Number** columns.

D. Remove the unique index from the **PhoneNumberID** column.

E. Change the PRIMARY KEY constraints to nonclustered indexes.

F. Execute on UPDATE STATISTICS FULLSCAN ALL statements in SQL Query Analyzer.

Answer: A, B.
Explanation: The query searches the City column and then references the LastName column for values that begin with 'W', thus the City column should be indexed. In addition, changing the PRIMARY KEY constraint, which is used to identify rows in a column, to use the LastName column before the FirstName column would improve query time performance, as the FirstName column does not have to be queried. Instead, the rows where

the FirstName column are in the same row as the rows in the LastName column that match the search criteria only need to be returned. The code for this search would be something like:

```
SELECT FirstName, LastName, PhoneNumberID
FROM PhoneNumbers
WHERE City=[certain city]
AND LastName LIKE 'W%'
```

Incorrect answers:

C: The AreaCode, Exchange, and Number columns are not referenced in by the query, thus the presence or lack of indexes on these columns would not affect query time performance.

D: The PhoneNumberID column is not referenced by the query, thus the presence or lack of indexes on this column would not affect query time performance.

E: Generall,y clustered indexes are faster that non-clustered indexes. The only reason to remove the clustered index on the primary key would be with the idea of using the clustered index on another column. There is no such option in this scenario, which makes the change from a clustered index to a non-clustered index pointless.

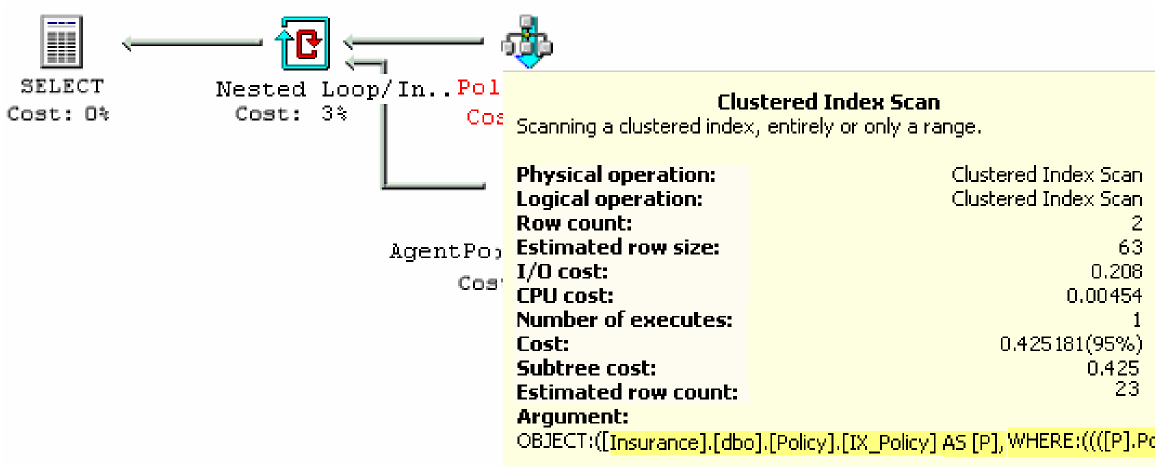
Note: In a nonclustered index the data is stored in one place, and the index in another, with pointers that link the index to the storage location of the data. The items in the index are stored in the order of the index key values, but the information in the table is stored in a different order. SQL Server 2000 searches for a data value by searching the nonclustered index to find the location of the data value in the table and then retrieves the data directly from that location. This makes nonclustered indexes the optimal choice for exact match queries because the index contains entries describing the exact location in the table of the data values being searched for in the queries. Thus, nonclustered indexes should be used on columns that contain a large number of distinct values, and in queries that do not return large result sets, and on columns that are frequently involved in search conditions of a query that return exact matches. A clustered index alters the physical storage order of the data in the table; as a result, a table can contain only one clustered index. However, the index can comprise multiple columns. A clustered index is particularly efficient on columns that are often searched for ranges of values. After the row with the first value is found using the clustered index, rows with subsequent indexed values are guaranteed to be physically adjacent. If a column is frequently used to sort the data retrieved from a table, it can be advantageous to cluster the table on that column to save the cost of a sort each time the column is queried.

F: SQL Server keeps statistics about the distribution of the key values in each index and uses these statistics to determine which index(es) to use in query processing. Users can create statistics on non-indexed columns by using the CREATE STATISTICS statement. However, query optimization depends on the accuracy of the distribution of the index. UPDATE STATISTICS should be re-run when there is significant change in the key values in the index on that index as occurs when a large amount of data in an indexed column has been added, changed, or removed, i.e., if the distribution of key values has changed, or the table has been truncated using the TRUNCATE TABLE statement and then repopulated. The STATS_DATE function can be used to check when the statistics were last updated. In this scenario large amounts of data have not been added, changed, or removed in the indexed column.

Q. 33
You are a database developer for an insurance company. You are tuning the performance of queries in SQL Query Analyzer. In the query pane, you create the following query:

```
SELECT P.PolicyNumber, P.IssueState, AP.Agent
FROM Policy AS P JOIN AgentPolicy AS AP
ON (P.PolicyNumber = AP.PolicyNumber)
WHERE IssueState = ‘IL’
AND PolicyDate BETWEEN ‘1/1/2000’ AND ‘3/1/2000’
AND FaceAmount > 1000000
```

You choose “Display estimated execution plan” from the query menu and execute the query. The query execution plan that is generated is shown in the estimated execution plan exhibit.



Clustered Index Scan
Scanning a clustered index, entirely or only a range.

Physical operation:	Clustered Index Scan
Logical operation:	Clustered Index Scan
Row count:	2
Estimated row size:	63
I/O cost:	0.208
CPU cost:	0.00454
Number of executes:	1
Cost:	0.425181(95%)
Subtree cost:	0.425
Estimated row count:	23

Argument:
OBJECT:([Insurance].[dbo].[Policy].[IX_Policy] AS [P], WHERE:((([P].Po

What should you do?

A. Rewrite the query to eliminate BETWEEN keyword.
B. Add a join hint that includes the HASH option to the query.
C. Add the WITH (INDEX(0)) table hint to the **Policy** table.
D. Update statistics on the **Policy** table.
E. Execute the DBCC DBREINDEX statement on the **Policy** table.

Answer: D.
Explanation: Updating the statistics in the Policy table could make the Query optimizer use the indexes in a better way. This would increase performance.

Note 1: Unexpectedly long-lasting queries and updates can be caused by:

- ∉ Slow network communication.
- ∉ Inadequate memory in the server computer.
- ∉ Lack of useful statistics.
- ∉ Out-of-date statistics.
- ∉ Lack of useful indexes.
- ∉ Lack of useful data striping.

Note 2: An index scan is used. An index seek would be faster.

Reference:
BOL, Query Tuning
INF: Troubleshooting Slow-Running Queries on SQL Server 7.0 or Later (Q243589)
BOL, Understanding Hash Joins
BOL, Table hints

Incorrect answers:
A: The syntax of the BETWEEN clause is correct and will return values that are true for the test expression, i.e. policy dates that are greater than or equal to 1/1/2000 and less than or equal to 3/1/2000. There is no way to rewrite the query, keep the result set, and improve the performance.

Note: The BETWEEN operator specifies a range to test. The result of this test is returned as a Boolean value, i.e. TRUE, FALSE or UNKNOWN. The value returned id TRUE if the test expression is greater than or equal to the first expression and less than or equal second expression, and UNKNOWN if any input to the BETWEEN predicate is NULL. The test expression, the first expression and the second expression must all be of the same data type. The correct syntax for the BETWEEN operator is:

<test expression> BETWEEN <first expression> AND <second expression>

B: The exhibit indicates that a nested loop is used to join the tables, and that a clustered index scan is on the AgentPolicy table. A hash join could improve the performance of the join and eliminate the clustered index scan. The query could run faster.

C: The estimated execution plan shows that a clustered index scan is going to be used, not a seek. It would be better if the query used an index seek instead. We could try to force an index seek with a table hint. However, we should use the hint index(1), not the hint index(0).
Index(0): If a clustered index exists, INDEX(0) forces a clustered index scan. But the query is already using a clustered index scan, so this would be no improvement.

Index(1): If a clustered index exists, INDEX(1) forces a clustered index scan or seek. This could improve the performance, but it is not listed as an option.

E: Rebuilding an index could improve performance of a query that uses that index. This is particularly true if the index is fragmented.

Note: DBCC DBREINDEX rebuilds all indexes defined for a table dynamically. In so doing, indexes enforcing either PRIMARY KEY or UNIQUE constraints can be rebuilt without having to drop and re-create those constraints. This means an index can be rebuilt without knowing the table's structure or constraints, which could occur after a bulk copy of data into the table. DBCC DBREINDEX can also rebuild all of the indexes for a table in one statement, which is easier than coding multiple DROP INDEX and CREATE INDEX statements. Because the work is done by one statement, DBCC DBREINDEX is automatically atomic, while individual DROP INDEX and CREATE INDEX statements would have to be put in a transaction to be atomic. Furthermore, DBCC DBREINDEX has a greater level of optimization than can be achieved with individual DROP INDEX and CREATE INDEX statements.

Q. 34
You are a database developer for a SQL Server 2000 database. You are planning to add new indexes, drop some indexes, and change other indexes to composite and covering indexes.

For documentation purposes, you must create a report that shows the indexes used by queries before and after you make changes. What should you do?

A. Execute each query in SQL Query Analyzer, and use the SHOWPLAN_TEXT option. Use the output for the report.

B. Execute each query in SQL Query Analyzer, and use the Show Execution Plan option. Use output for the report.

C. Run the Index Tuning Wizard against a Workload file. Use the output for the report.

D. Execute the DBCC SHOW_STATISTICS statement. Use the output for the report.

Answer: A.
Explanation: The SHOWPLAN_TEXT command causes SQL Server not to execute Transact-SQL statements. Instead, SQL Server returns detailed information about how the statements are executed. The textual output is suitable for reports used in documentation.

Note: SQL Query Analyzer is a graphical user interface that can be used to design and test Transact-SQL statements, batches, and scripts interactively. SQL Query Analyzer can be called from SQL Server Enterprise Manager and offers a graphical diagram of the **showplan** information showing the logical steps built into the execution plan of a Transact-SQL statement. This allows programmers to determine what specific part of a poorly performing query is using a lot of resources. Programmers can then explore changing the query in ways that minimize the resource usage while still returning the correct data. The results can be output in a text format.

Incorrect answers:

B: Show Execution Plan would show a graphical view of the execution plan. A textual presentation would be more suitable for a documentation report.

C: The Index Tuning Wizard is used for analyzing and tuning indexes. It cannot be used to make reports on specific queries.

Note: 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.

D: DBCC SHOW_STATISTICS returns results that display the current distribution statistics for the specified target on the specified table. The results returned indicate the selectivity of an index and provide the basis for determining whether or not an index is useful to the query optimizer. The results returned are based on the distribution steps of the index. The results are graphical, not textual.

Q. 35
You are a database developer for a hospital. You are designing a SQL Server 2000 database that will contain physician and patient information. This database will contain a table named Physicians and a table named Patients.

Physicians treat multiple patients. Patients have a primary physician and usually have a secondary physician. The primary physician must be identified as the primary physician. The Patients table will contain no more than 2 million rows.

You want to increase I/O performance when data is selected from the tables. The database should be normalized to the third normal form.

Which script should you use to create the tables?

A.
```
CREATE TABLE Physicians
(
Physicians ID int NOT NULL CONSTRAINT PK_Physicians PRIMARY KEY CLUSTERED,
LastName varchar(25) NOT NULL,
)
GO
CREATE TABLE Patients
(
PatientID bigint NOT NULL CONSTRAINT PK_Patients PRIMARY KEY CLUSTERED,
LastName varchar (25) NOT NULL,
FirstName varchar (25) NOT NULL,
PrimaryPhysician int NOT NULL,
SecondaryPhysician int NOT NULL,
CONSTRAINT PK_Patients_Physicians1 FOREIGN KEY (PrimaryPhysician) REFERENCES
Physicians (PhysicianID),
CONSTRAINT PK_Patients_Physicians2 FOREIGN KEY (SecondaryPhysician) REFERENCES
Physicians (PhysicianID)
)
```

B.
```
CREATE TABLE Patients
(
PatientID smallint NOT NULL CONSTRAINT PK_Patients PRIMARY KEY CLUSTERED,
LastName varchar(25) NOT NULL,
FirstName varchar (25) NOT NULL,
PrimaryPhysician int NOT NULL,
SecondaryPhysician int NOT NULL,
)
GO
CREATE TABLE Physicians
(
PhysicianID smallint NOT NULL CONSTRAINT PK_Physicians PRIMARY KEY CLUSTERED,
LastName varchar (25) NOT NULL,
FirstName varchar (25) NOT NULL,
CONSTRAINT PK_Physicians_Patients FOREIGN KEY (PhysicianID) REFERENCES Patients
(PatientID)
)
```

C.
```
CREATE TABLE Patients
(
PatientID bigint NOT NULL CONSTRAINT PK_Patients PRIMARY KEY CLUSTERED,
LastName varchar (25) NOT NULL,
FirstName varchar (25) NOT NULL,
)
GO
CREATE TABLE Physicians
(
PhysicianID int NOT NULL CONSTRAINT PK_Physician PRIMARY KEY CLUSTERED,
LastName varchar (25) NOT NULL,
FirstName varchar (25) NOT NULL,
)
GO
CREATE TABLE PatientPhysician
(
PatientPhysicianID bigint NOT NULL CONSTRAINT PK_PatientsPhysicians PRIMARY KEY
CLUSTERED,
PhysicianID int NOT NULL,
PatientID bigint NOT NULL,
PrimaryPhysician bit NOT NULL,
FOREIGN KEY (PhysicianID) REFERENCES Physicians (PhysicianID),
FOREIGN KEY (PatientID) REFERENCES Patients (PatientID)
)
```

D.

```
CREATE TABLE Patients
(
PatientID int NOT NULL PRIMARY KEY,
LastName varchar (25) NOT NULL,
FirstName varchar (25) NOT NULL,
)
GO
CREATE TABLE Physicians
(
PhysicianID int NOT NULL PRIMARY KEY,
LastName varchar (25) NOT NULL,
FirstName varchar (25) NOT NULL,
)
GO
CREATE TABLE PatientPhysician
(
PhysicianID int NOT NULL REFERENCES Physicians (PhysicianID),
PatientID int NOT NULL REFERENCES Patients (PatientID), PrimaryPhysician bit NOT NULL,
CONSTRAINT PK_PatientsPhysicians PRIMARY KEY (PhysicianID, PatientID)
)
```

Answer: D.
Explanation: We want to normalize the design. We notice that there is a many-to-many relationship between the Patients and the Physicians tables. One patient can have many doctors. (“Two” counts as “many” in logical database design.) Doctors can have many patients. So we have to create a third table, PatientPhysician to connect them through two one-to-many relations, since SQL Server doesn’t support many-to-many relations.

We must carefully choose the datatype of the primary keys. No table would require more than about 2 million rows. We can safely use the int datatype for the primary key columns in all the tables, since the maximum value of an int column is 2,147,483,647.

We must also take care in creating foreign keys referencing the primary keys in the appropriate tables. We must create two foreign keys in the PatientPhysician table: one referencing the Patient table and one referencing the Physicians table. It is also important that the corresponding columns have the same datatype, in this scenario the int datatype.

Note: int datatype
The int datatype represents whole numbers -2,147,483,648 through 2,147,483,647 and uses a storage size of 4 bytes, while the smallint ranges from -32,768 through 32,767 and the tinyint ranges from 0 through 255. It is recommended that the smallest data type that provides a sufficient range of values be used.

Bigint can handle integers in the range of 2^63 (-9223372036854775808) through 2^63-1 (9223372036854775807).

Incorrect answers:
A: This code does not allow the column holding the secondary physician identifier, i.e. the SecondaryPhysician column in the Patient table, to hold nulls. This will create database integrity problems when the patient does not have a secondary physician. Furthermore, the database is not normalized to the third normal form, as PrimaryPhysician and SecondaryPhysician columns in the Patient table are not dependent upon the PRIMARY KEY placed on the PatientID column.

B: This code does not allow the column holding the secondary physician identifier, i.e. the SecondaryPhysician column in the Patient table, to hold nulls. This will create database integrity problems when the patient does not have a secondary physician. This code also uses the smallint data type to uniquely identify the patients in the PatientID column in the Patient table. As this data type has a range of -32,768 through 32,767, it can only apply unique identifiers to 65, 535 patients, which would be insufficient since the hospital has more than 2 million patients. Furthermore, the database is not normalized to the third normal form, as PrimaryPhysician and SecondaryPhysician columns in the Patient table are not dependent upon the PRIMARY KEY placed on the PatientID column.

C: There is no need to use the bigint datatype. A column with the int datatype would be able to hold more than 2000 million distinct values, and only around 2 million rows are required. We should not use a larger datatype than necessary.

Q. 36
You are the database developer for your company's SQL Server 2000 database. This database contains a table named Invoices. You are a member of the db_owner role.

Eric, a member of the HR database role, created the Trey_Research_UpdateInvoices trigger on the Invoices table. Eric is out of the office, and the trigger is no longer needed. You execute the following statement in the Sales database to drop the trigger:

DROP TRIGGER Trey_Research_UpdateInvoices

You receive the following error message:

```
Cannot drop the trigger 'Trey_Research_UpdateInvoices', because it does not
exist in the system catalog.
```

What should you do before you can drop the trigger?

A. Add your login name to the **HR database** role.

B. Qualify the trigger name with the trigger owner in the DROP TRIGGER statement.

C. Disable the trigger before executing the DROP TRIGGER statement.

D. Define the trigger number in the DROP TRIGGER statement.

E. Remove the text of the trigger from the **sysobjects** and **syscomments** system tables.

Answer: B.
Explanation: By default, permissions to delete a trigger rest with the owner of the table upon which the trigger is defined. However, members of the db_owner and db_ddladmin fixed database role or sysadmin fixed server role can implement the DROP TRIGGER statement by explicitly specifying the owner in the DROP TRIGGER statement. As you are a member of the db_owner fixed database role, you may specify the owner in the DROP TRIGGER statement.

Incorrect answers:
A: Adding your login name to the HR database role in order to perform the DROP TRIGGER statement would not allow you to drop the trigger, as the ownership of a database object does not reside with a group but with the creator of the database object.

C: Disabling a trigger will not permit that trigger to be deleted, as permission to delete a trigger rests with the owner of the table upon which the trigger is defined.

D: Triggers are identified by a unique name in MS SQL 2000, and not by an assigned number. These unique names should be specified when making reference to a trigger.

E: System tables are critical to the operation of SQL Server 2000. It is recommended that users should not directly alter system tables, as this could lead to data loss or could result in an instance of SQL Server 2000 not being able to run.

Q. 37
You have designed the database for a web site that is used to purchase concert tickets. During a ticket purchase, a buyer views a list of available tickets, decides whether to buy the tickets, and then attempts to purchase the tickets. This list of available tickets is retrieved in a cursor.

For popular concerts, thousands of buyers might attempt to purchase tickets at the same time. Because of the potentially high number of buyers at any one time, you must allow the highest possible level of concurrent access to the data.

How should you design the cursor?

A. Create a cursor within an explicit transaction, and set the transaction isolation level to REPEATABLE READ.

B. Create a cursor that uses optimistic concurrency and positioned updates. In the cursor, place the positioned UPDATE statements within an explicit transaction.

C. Create a cursor that uses optimistic concurrency. In the cursor, use UPDATE statements that specify the key value of the row to be updated in the WHERE clause, and place the UPDATE statements within an implicit transaction.

D. Create a cursor that uses positioned updates. Include the SCROLL_LOCKS argument in the cursor definition to enforce pessimistic concurrency. In the cursor, place the positioned UPDATE statements within an implicit transaction.

Answer: B.
Explanation: A good suggestion is to use a combination of normal and positioned cursors. The “WHERE CURRENT OF” option is the critical part of our update script, and it is used in the cursor to update our data one row at a time in a transaction loop. This would keep the locking localized to the current row and performance would be improved. Optimistic locking with explicit transaction is the best method for positioned updates.

Note: Operations in a relational database act on a complete result set. A result set is returned by a SELECT statement and consists of all the rows that satisfy the conditions in the WHERE clause of the statement. However, applications such as interactive online applications cannot always work effectively with the entire result set as a unit. These applications need a mechanism to work with a small block of rows at a time. Cursors are an extension to result sets that provide this mechanism. They extend result processing by allowing positioning at specific rows of the result set; retrieving a small block of rows from the current position in the result set; supporting data modifications to the rows at the current position in the result set; supporting different levels of visibility to changes made by other users to the database data that is presented in the result sets; and providing Transact-SQL statements in scripts, stored procedures, and triggers access to the data in a result set. In situation where multiple users will simultaneously access a table, concurrent access will be required. However, with concurrent access, data soon becomes unreliable without some type of control over the concurrent access. The standard cursors control concurrent access through several options. These controls are pessimistic or optimistic. Pessimistic concurrency controls are of two types: read-only, which does not permit data updates, and scroll-locked, in which rows are locked when they are fetched inside a user-initiated transaction and no other user can gain access to those rows until the locks are released. Optimistic concurrency control which does not place locks on the database, allowing multiple users to access the same rows, and must resolve any update conflicts that may occur as a result of simultaneous updates.

Updateable cursors support data modification statements that update rows through the cursor through position updates. When positioned on a row in an updateable cursor, update or delete operations that target the base table rows used to build the current row in the cursor can be performed. The positioned updates are performed on the same connection that opened the cursor. This allows the data modifications to share the same transaction space as the cursor, and prevents the updates from being blocked by locks held by the cursor. Positioned updates in a cursor can be performed through Transact-SQL by using the WHERE CURRENT OF clause on an UPDATE statement and can be placed within explicit transactions. With explicit transactions, both the start and end of the transaction can be specified. Transact-SQL uses the BEGIN TRANSACTION, COMMIT TRANSACTION, and ROLLBACK TRANSACTION statements to define explicit transactions. BEGIN TRANSACTION indicates the starting point of an explicit transaction, COMMIT TRANSACTION is used to end a transaction if no errors were encountered, and ROLLBACK TRANSACTION is used to erase a transaction in which errors are encountered.

Incorrect answers:

A: Selecting the REPEATABLE READ transaction isolation level, the highest level, would decrease concurrency and increase locking. This would be contrary to the requirements of this scenario.

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.

C: Positioned updates in a cursor can be performed through Transact-SQL by using the WHERE CURRENT OF clause on an UPDATE statement and can be placed within explicit transactions. This solution however does not refer to the WHERE clause.

D: Implicit transactions would not be able to keep the locking only to the current row in the cursor. Locking would increase and performance would suffer. It is better to use an explicit cursor in conjunction with the CURRENT OF construct.

Q. 38
You are a database developer for a company that conducts telephone surveys of consumer music preferences. As the survey responses are received from the survey participants, they are inserted into a table named SurveyData. After all of the responses to a survey are received, summaries of the results are produced.

You have been asked to create a summary by sampling every fifth row of responses for a survey. You need to produce the summary as quickly as possible.

What should you do?

A. Use a cursor to retrieve all of the data for the survey.
Use FETCH RELATIVE 5 statement to select the summary data from the cursor.

B. Use a SELECT INTO statement to retrieve the data for the survey into a temporary table.
Use a SELECT TOP 1 statement to retrieve the first row from the temporary table.

C. Set the query rowcount to five. Use a SELECT statement to retrieve and summarize the survey data.

D. Use a SELECT TOP 5 statement to retrieve and summarize the survey data.

Answer: A.
Explanation: The FETCH RELATIVE 5 clause returns the 5th row in the cursor from the current row and makes that row the new current row. It is thus possible to use FETCH RELATIVE 5 to return every 5th row in the cursor. Note the FETCH clause takes a variable expression. Thus FETCH RELATIVE 3 will return every 3rd row in the cursor and FETCH RELATIVE 7 every 7th row.

Incorrect answers:
B: The SELECT TOP clause limits the number of rows returned in the result set. SELECT TOP 1 specifies that only the first row should be output from the query result set.
C: The SET ROWCOUNT clause was used prior to SQL Server 7 to limit SQL Server to sending only a specified number of rows to the client. Every SELECT statement would stop sending rows back to the client after the specified number of rows had been sent. This SET ROWCOUNT clause is the

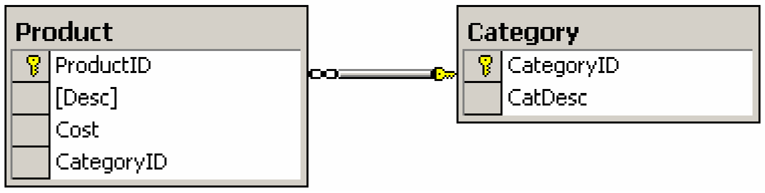
functionally equivalent to the SELECT TOP clause and so SET ROWCOUNT 5 is identical to SELECT TOP 5. However, the SELECT TOP clause is more efficient than the SET ROWCOUNT clause.

D: The SELECT TOP clause limits the number of rows returned in the result set. SELECT TOP 5 specifies that only the first 5 rows should be output from the query result set.

Q. 39
You are a database developer for a lumber company. You are performing a one-time migration from a flat-file database to SQL Server 2000. You export the flat-file database to a text file in comma-delimited format. The text file is shown in the Import file exhibit button.

```
1111, '*4 Interior', 4, 'Interior Lumber', 1.12
1112, '2*4 Exterior', 5, 'Exterior Lumbar', 1.87
2001, '16d galvanized',2, 'Bulk Nails', 2.02
2221, '8d Finishing brads',3, 'Nails', 0.01
```

You need to import this file into SQL Server tables named Product and Category. The product and category tables are shown in the product and Category Tables exhibit.



You want to import the data using the least amount of administrative effort. What should you do?

A. Use the **bcp** utility, and specify the **–t** option.

B. Use the BULK INSERT statement, and specify the FIRE_TRIGGERS argument.

C. Use the SQL-DMO **BulkCopy2** object and set the **TableLock** property to TRUE.

D. Use Data Transformation Services to create two Transform Data tasks.
For each task, map the text file columns to the database columns.

Answer: D.
Explanation: In this scenario the data in the text file must be imported into two SQL Server 2000 tables. For this to be accomplished, the data in the relevant text file column must be mapped to the columns in the correct table. This can only be accomplished by using Data Transformation Services to create two Transform Data tasks, one for each table.

Incorrect answers:
A: The bcp utility is used to copy data between a SQL Server 2000 database and a data file in a user-specified format, while the –t option specifies the field terminator that should be used. It can only copy data between single table and text file sources.

B: The BULK INSERT task provides the quickest way to copy large amounts of data into a SQL Server table or view. To ensure high-speed data movement, transformations cannot be performed on the data while it is moved from the source file to the table or view. This requires that the text file columns match the columns in the destination table. Consequently, the BULK INSERT statement can only insert data into one table that matches the source table. Furthermore, all bulk copy operations support the bulk copy hint, FIRE_TRIGGERS. If FIRE_TRIGGERS is specified on a bulk copy operation that is copying rows into a table, INSERT and INSTEAD OF triggers defined on the destination table are executed for all rows inserted by the bulk copy operation. BULK INSERT can only copy data between single table and text file sources.

C: BulkCopy2 is used as the parameters to the ImportData method of the Table object and the ExportData method of the Table and View of a single bulk copy command issued against a SQL Server 2000 database, and it extends the functionality of BulkCopy for use with features that are new in SQL Server 2000. BulkCopy2 can only be used to copy data between single table and text file sources.

Q. 40
You are a database developer for a database named Accounts at Woodgrove Bank. A developer is creating a multi-tier application for the bank. Bank employees will use the application to manage customer accounts. The developer needs to retrieve customer names from the accounts database to populate a drop-down list box in the application. A user of the application will use the list box to locate a customer account.

The database contains more than 50,000 customer accounts. Therefore, the developer wants to retrieve only 25 rows as the user scrolls through the list box. The most current list of customers must be available to the application at all times.

You need to recommend a strategy for the developer to use when implementing the drop-down list box. What should you recommend?

A. Create a stored procedure to retrieve all of the data that is loaded into the list box.

B. Use an API server-side cursor to retrieve the data that is loaded into list box.

C. Retrieve all of the data at once by using a SELECT statement, and then load the data into the list box.

D. Use a Transact-SQL server-side cursor to retrieve the data that is loaded into the list box.

Answer: B.
Explanation: Using an API server side cursor, the result set is located at the client, not at the server. This method is the most efficient way to retrieve rows from the server.

The OLE DB, ODBC, ADO, and DB-Library API support the mapping of cursors over the result sets of executed SQL statements. The SQL Server OLE DB provider, SQL Server ODBC driver, and DB-Library dynamic-link library implement these operations through the use of API server cursors. API server cursors are implemented on the server and managed by API cursor functions. As the application calls the API cursor functions, the cursor operation is transmitted to the server by the OLE DB provider, ODBC driver, or DB-Library DLL. When using an API server cursor in OLE DB, ODBC, and ADO, the functions or methods of the API should be used to open a connection, to set attributes or properties defining the characteristics of the cursor the API automatically maps over each result set, to execute one or more Transact-SQL statements, and to use API functions or methods to fetch the rows in the result sets.

Incorrect answers:
A: A stored procedure could not, by itself, scroll through specific rows. We must use a cursor to accomplish this.

Note: When applications are created with SQL Server 2000, the Transact-SQL programming language is the primary programming interface between the applications and the SQL Server database. When using Transact-SQL programs, there are two methods for storing and executing the programs: they can be stored locally and can create applications that send the commands to SQL Server and process the results, or they can be stored as stored procedures in SQL Server and can create applications that execute the stored procedures and process the results. Stored procedures in SQL Server can accept input parameters and return multiple values in the form of output parameters to the calling procedure or batch. They contain programming statements that perform operations in the database, including calling other procedures, and they return a status value to a calling procedure or batch to indicate success or failure (and the reason for failure). Although stored procedures cannot be used directly in an expression, they are more beneficial than Transact-SQL programs stored locally on client computers, as they allow modular programming, allow faster execution, can reduce network traffic and can be used as a security mechanism.

C: The SELECT statement returns the rowset of the data that meets the conditions set in its FROM and WHERE clauses. SELECT statements do not perform the same function as cursors, which return small blocks of rowsets and allow the user to scroll through the rest of the returned rowsets as the user requires them.

D: A server-side cursor stores the result set on the server, while an API server-side cursor stores the result set on the client. This means a server-side cursor require more resources of the SQL Server computer; in particular, RAM would have to be allocated for the server-side cursor. Therefore, an API server-side cursor is preferred in this scenario.

Note: Server-side cursors do not support all Transact-SQL statements. They do not support Transact-SQL statements that generate multiple result sets, and cannot therefore be used when the application executes a stored procedure or a batch that contains more than one SELECT statement. Server cursors also do not support SQL statements that contain the keywords COMPUTE, COMPUTE BY, FOR BROWSE, or INTO.

Q. 41
You are a database developer for Litware, Inc. You are restructuring the company's sales database. The database contains customer information in a table named Customers. This table includes a character field named Country that contains the name of the country in which the customer is located.

You have created a new table named Country. The scripts that were used to create the Customers and Country tables are shown in the exhibit.

```
CREATE TABLE dbo.Country
     (
     CountryID int IDENTITY(1,1) NOT NULL,
     CountryName char(20) NOT NULL,
     CONSTRAINT PK_Country PRIMARY KEY CLUSTERED (CountryID)
     )
CREATE TABLE dbo.Customers
     (
     CustomerID int NOT NULL,
     CustomerName char(30) NOT NULL,
     Country char(20) NULL,
     CONSTRAINT PK_Customers PRIMARY KEY CLUSTERED (CustomerID)
     )
```

You must move the country information from the Customers table into the new Country tables as quickly as possible.

Which script should you use?

A.
```
INSERT INTO Country (CountryName)
SELECT DISTINCT Country
FROM Customers
```

B.
```
SELECT (*) AS ColID, c1.Country
INTO Country
FROM (SELECT DISTINCT Country FROM Customers)AS c1,
        (SELECT DISTINCT Country FROM Customers) AS c2,
WHERE c1.Country >=c2.Country
GROUP BY c1.Country ORDER BY 1
```

C.
```
DECLARE @Country char (20)

DECLARE cursor_country CURSOR
        FOR SELECT Country FROM Customers
OPEN cursor_country
FETCH NEXT FROM cursor_country INTO @Country

WHILE (@@FETCH_STATUS <> -1)
BEGIN
        If NOT EXISTS (SELECT CountryID
                                FROM Country
                                WHERE CountryName = @Country)
INSERT INTO Country (CountryName) VALUES (@Country)
FETCH NEXT FROM cursor_country INTO @Country
END

CLOSE cursor_country
DEALLOCATE cursor_country
```

D.
```
DECLARE @SQL varchar (225)
SELECT @SQL = ‘bcp “SELECT ColID = COUNT(*), c1. Country’ +
        ‘FROM (SELECT DISTINCT Country FROM Sales..Customers) AS
c1, ' +
        (SELECT DISTINCT Country FROM Sales..Customers) AS c2 '
+
        WHERE c1.Country >= c2.Country’ +
        ‘GROUP BY c1.Country ORDER BY 1’ +
        ‘query out c:\country.txt -c’
EXEC master..xp_cmdshell @SQL, no_output
EXEC master..xp_cmdshell ‘bcp Sales..Country in c:\country. Txt-c’, no_output
```

Answer: C.
Explanation:
The code using the cursor looks complicated but it is the only solution that handles the NULL values in the Country column in the Customers table. It will skip all NULL rows in the Customers table.

Note: The code works like this:
The cursor is declared as the Country column of the Customers table

```
DECLARE @Country char (20)
DECLARE cursor_country CURSOR
        FOR SELECT Country FROM Customers
```

Then the cursor is set to it initial value.

```
OPEN cursor_country
```

The cursor will insert the rows one by one with the loop construction WHILE (@@FETCH_STATUS <> -1).
The statement

```
If NOT EXISTS (SELECT CountryID
                                FROM Country
                                WHERE CountryName = @Country)
```

checks if the cursor, the Country column of the Customers table, contains a new country. If not, the country already exists, or if there is a NULL value, the row is skipped.

If the cursor contains a new country then the row is inserted into the Country table:

```
INSERT INTO Country (CountryName) VALUES (@Country)
```

In the end of the loop the cursor fetches the next Country column value:

```
FETCH NEXT FROM cursor_country INTO @Country
```

Then the loop is ended:

```
END
```

Incorrect answers:
A: The simplicity of this suggested solution is tempting. This simple code will not handle NULL values in the country column in the Customers table. In this is the case the code will fail to execute.

Note: The INSERT INTO statement is used to insert rows into an existing table. The INSERT INTO can basically be used in two ways:

∉ to insert single values by the statement INSERT INTO *tablename* VALUES *listofvalues*

∉ to insert multiple values by the statement INSERT INTO *tablename SELECT statement*

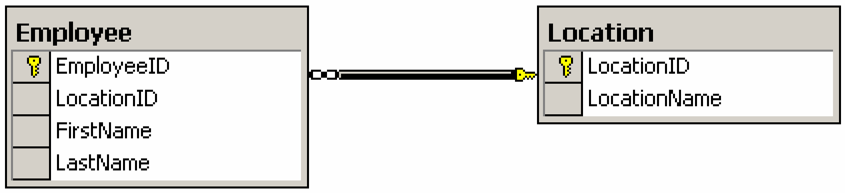
We insert all distinct rows from the Customers table into the Country table so that the same country isn't stored twice. There is a catch here: if there were some null rows in customer table, the statement would fail to execute.

B: The SELECT INTO statement creates a new table, but we already have a table and we want to copy rows to the existing table.

D: This solution is complicated and inefficient.

Q. 42

You are a database developer for Contoso, Ltd. The company has a database named HumanResources that contains information about all employees and office locations. The database also contains information about potential employees and office locations. The tables that contain this information are shown in the exhibit.



Current employees are assigned to a location, and current locations have one or more employees assigned to them. Potential employees are not assigned to a location, and potential office locations do not have any employees assigned to them.

You need to create a report to display all current and potential employees and office locations. The report should list each current and potential location, followed by any employees who have been assigned to that location. Potential employees should be listed together.

Which script should you use?

A.
```
SELECT l.LocationName, e.FirstName, e.LastName
FROM Employee AS e LEFT OUTER JOIN Location AS 1
    ON e.LocationID= l.LocationID
ORDER BY l.LocationName, e.LastName, e.FirstName
```

B.
```
SELECT l.LocationName, e.FirstName, e.LastName
FROM Location AS 1 LEFT OUTER JOIN EMPLOYEE AS 1
    ON e.LocationID= l.LocationID
ORDER BY l.LocationName, e.LastName, e.FirstName
```

C.
```
SELECT l.LocationName, e.FirstName, e.LastName
FROM Employee AS e FULL OUTER JOIN Location AS 1
    ON e.LocationID= l.LocationID
ORDER BY l.LocationName, e.LastName, e.FirstName
```

D.
```
SELECT l.LocationName, e.FirstName, e.LastName
FROM Employee AS e CROSS JOIN Location AS 1
ORDER BY l.LocationName, e.LastName, e.FirstName
```

E.
```
SELECT l.LocationName, e.FirstName, e.LastName
FROM Employee AS e, Location AS 1
ORDER BY l.LocationName, e.LastName, e.FirstName
```

Answer: C.
Explanation: This scenario requires us to JOIN two tables in order to display all current and potential employees and office locations in such a way that each current and potential location is listed, followed by any employees who have been assigned to that location. Potential employees should be listed together. There are locations with no employees and employees without a location, so we are forced to use a FULL OUTER JOIN. A FULL OUTER JOIN accepts NULL values in the joining column from either table.

Note: By using joins, we can retrieve data from two or more tables based on logical relationships between the tables. Joins can be specified in either the FROM or WHERE clauses and can be categorized as INNER JOIN, OUTER JOIN, or CROSS JOIN. Inner joins use a comparison operator to match rows from two tables based on the values in common columns from each table. Outer joins can be a left, a right, or a full outer join and are specified with one of the following sets of keywords when they are specified in the FROM clause: LEFT OUTER JOIN, RIGHT OUTER JOIN, or FULL OUTER JOIN. The result set of a left outer join includes all the rows from the left table specified in the LEFT OUTER clause, not just the ones in which the joined columns match. When a row in the left table has no matching rows in the right table, the associated result set row contains null values for all select list columns coming from the right table. A right outer join is the reverse of a left outer join. A full outer join returns all rows in both the left and right tables. Any time a row has no match in the other table, the select list columns from the other table contain null values. When there is a match between the tables, the entire result set row contains data values from the base tables. Finally, cross joins return all rows from the left table; each row from the left table is combined with all rows from the right table. Cross joins are also called Cartesian products. As we need to list the columns in such a way that each current and potential location is listed, followed by any employees who have been assigned to that location. A FULL OUTER JOIN is required. But we also need to list potential employees together. This can be achieved through the use of the ORDER BY clause.

Incorrect answers:

A: All locations must be included in the result set, but a LEFT OUTER join would not list locations that have no employees.

Note: The result set of a LEFT OUTER JOIN includes all the rows from the left table specified in the LEFT OUTER clause, not just the ones in which the joined columns match. When a row in the left table has no matching rows in the right table, the associated result set row contains null values for all select list columns coming from the right table.

B: All employees must be included in the result set, but a RIGHT OUTER join would not list employees that have no location.

Note: The result set of a RIGHT OUTER JOIN includes all the rows from the right table specified in the RIGHT OUTER clause, not just the ones in which the joined columns match. When a row in the right table has no matching rows in the left table, the associated result set row contains null values for all select list columns coming from the left table.

D: The result set of a CROSS JOIN includes all rows from the left table, which are combined with all rows from the right table. But we only want to show the locations and their employees, not all possible location/employee combinations.

E: Not specifying any JOIN statement would result in a cross join, which is not the proper solution here.

Q. 43
You are designing a database for a web-based ticket reservation application. There might be 500 or more tickets available for any single event. Most users of the application will view fewer than 50 of the available tickets before purchasing tickets.

However, it must be possible for a user to view the entire list of available tickets. As the user scrolls through the list, the list should be updated to reflect that tickets have been sold to other users. The user should be able to select tickets from the list and purchase the tickets.

You need to design a way for the user to view and purchase available tickets. What should you do?

A. Use a scrollable static cursor to retrieve the list of tickets.
Use positioned updates within the cursor to make purchases.

B. Use a scrollable dynamic cursor to retrieve the list of tickets.
Use positioned updates within the cursor to make purchases.

C. Use a stored procedure to retrieve the list of tickets.
Use a second stored procedure to make purchases.

D. Use a user-defined function to retrieve the list of tickets.
Use a second stored procedure to make purchases.

Answer: B.
Explanation: In dynamic cursors, membership is not fixed – subsequent fetches might include newly qualifying rows, or previously qualifying rows might disappear, if changes have been made within the cursor or if changes have been made by others. If the cursor has not locked the rows of the result set, the changes made by others are seen in the cursor only on a subsequent the fetch on the cursor. Thus, dynamic cursors are updateable. Furthermore, updateable cursors support data modification statements that update rows through the cursor. When positioned on a row in an updateable cursor, update or delete operations can be performed. These operations target the base table rows used to build the current row in the cursor. These are called position updates. The positioned updates are performed on the same connection that opened the cursor. This allows the data modifications to share the same transaction space as the cursor, and prevents the updates from being blocked by locks held by the cursor.

Incorrect answers:
A: When using a static cursor, the user operates in private on a temporary copy of the data that qualifies for the cursor. This temporary copy of the qualifying data is stored in tempdb and is set to read-only in the temporary database. The rows in the cursor and the data values of the rows cannot change when data is fetched anywhere in the cursor. Thus the data contained in the temporary database that the static cursor accesses cannot be updated. Furthermore, membership in a static cursor is fixed—that is, as when data is fetched in any direction, new rows cannot be added to the temporary database.

C: To meet the requirements of this scenario we must use a cursor.

Note: Operations in a relational database act on a complete result set. A result set is returned by a SELECT statement and consists of all the rows that satisfy the conditions in the WHERE clause of the statement. However, applications such as interactive online applications cannot always work effectively with the entire result set as a unit. These applications need a mechanism to work with a small block of rows at a time. Cursors are an extension to result sets that provide this mechanism. This solution does not make use of the mechanism.

D: To meet the requirements of this scenario we must use a cursor.

Note: Operations in a relational database act on a complete result set. A result set is returned by a SELECT statement and consists of all the rows that satisfy the conditions in the WHERE clause of the statement. However, applications such as interactive online applications cannot always work effectively with the entire result set as a unit. These applications need a mechanism to work with a small block of rows at a time. Cursors are an extension to result sets that provide this mechanism. This solution does not make use of the mechanism.

Q. 44
You are a database consultant. You have been hired by a local dog breeder to develop a database. This database will be used to store information about the breeder's dogs. You create a table named Dogs by using the following script:

```
CREATE TABLE[dbo].[Dogs]
     (
     [DogID]  [int] NOT NULL,
     [BreedID]  [int] NOT NULL,
     [DateofBirth]  [datetime] NOT NULL,
     [WeightAtBirth] [decimal] (5, 2) NOT NULL,
     [NumberOfSiblings]  [int] NULL,
     [MotherID] [int] NOT NULL,
     [FatherID] [int] NOT NULL
     ) ON [PRIMARY]
GO
     ALTER TABLE [dbo].[Dogs] WITH NOCHECK ADD
     CONSTRAINT [PK_Dogs]PRIMARY KEY CLUSTERED
     (
     [DogID]
     ) ON [PRIMARY]
GO
```

You must ensure that each dog has a valid value for the MotherID and FatherID columns. You want to enforce this rule while minimizing disk I/O.

What should you do?

A. Create an AFTER INSERT trigger on the **Dogs** table that rolls back the transaction of the **MotherID** or **FatherID** column is not valid.
B. Create a table-level CHECK constraint on the **MotherID** and **FatherID** columns.
C. Create two FOREIGN KEY constraints: one constraint on the **MotherID** column and one constraint on the **FatherID** column. Specify that each constraint reference the **DogID** column.
D. Create a rule and bind it to the **MotherID**. Bind the same rule to the **FatherID** column.

Answer: C.
Explanation: From logical database design reasoning we can see that there is a one-to-many relationship between the DogID column and the MotherID column:

- Every dog has one specific mother (and father).
- Every mother (or father) can have 0, 1, or several children (puppies).

One-to-many relations are implemented by foreign key constraints in most RDBMS like SQL Server. Two foreign key constraints, one for each column, can be used to ensure that a new row contains proper values for the MotherID and FatherID. Foreign keys constraints that reference a column in the same table, usually the primary key, are not unusual in relational database design. This is called self-referencing and is mostly used to define hierarchical structures, for example family (or dog!) relations like in this example.

Note: A FOREIGN KEY constraint is placed on a column or combination of columns to establish and enforce a referential integrity between the data in two tables in a database. A link is created between two tables by adding the column or columns that hold one table's PRIMARY KEY values or UNIQUE constraint to a second table. This column becomes a foreign key in the second table and can be created when that table is created or altered. A table can also be self-referencing – that is, the foreign key can reference one or more columns in the same table.

Incorrect answers:
A: A trigger can be constructed that accomplish the same thing as the preferred solution with foreign keys. But there are drawbacks with triggers compared to foreign keys:

1. They must be coded.
2. They are not as efficient as foreign key constraints. In particular a trigger for this scenario would require a **select dogs** clause and use some mechanism to decide if there are corresponding rows that match the MotherID and FatherID columns respectively.

Note on triggers:
SQL Server 2000 uses two primary mechanisms, constraints and triggers, to enforce business rules and data integrity. Triggers are a special class of stored procedures that execute automatically when an UPDATE, INSERT, or DELETE statement is issued against a table or view. It can reject or accept each data modification transaction as a whole. By using a correlated subquery in a trigger, such as an AFTER INSERT trigger, the trigger can examine the modified rows one by one. This will prevent a complete rollback of all data modifications when some of the data changes are not acceptable; instead, the trigger deletes only the rows containing unacceptable data.

B: One-to-many relations cannot be enforced by CHECK constraints.

Note: CHECK constraints are used to enforce domain integrity by checking that invalid values are not entered in a column. They are similar to FOREIGN KEY constraints in that they control the values that can be placed in a column. FOREIGN KEY constraints are able to get the list of valid values from another table, while CHECK constraints determine the valid values from a logical expression that is not based on data in another column or table. It is possible to apply multiple CHECK constraints to a single column; these are applied in the order in which they were created. It is also possible to apply a single CHECK constraint to multiple columns by creating them at the table level.

D: One-to-many relations cannot be enforced by rules.

Note: Rules are used in cases where backward compatibility is required. They perform the same function as CHECK constraints. CHECK constraints are the preferred, standard way to check or restrict the values in a given column. CHECK constraints are also more concise than rules: multiple CHECK constraints can be applied to a column, while only one rule can be applied per column. CHECK constraints are also specified as part of the CREATE TABLE statement while rules are created as separate objects and then bound to the column.

Q. 45
You are the database developer for a company that provides consulting service. The company maintains data about its employees in a table named Employee. The script that was used to create the Employee table is shown in the exhibit.

```
CREATE TABLE Employee
          (
          EmployeeID int NOT NULL;
          EmpType char (1) NOT NULL,
          EmployeeName char (50) NOT NULL,
          Address char (50) NULL,
          Phone char (20) NULL,
          CONSTRAINT PK_Employee PRMARY KEY (Employee ID)
          )
```

The EmpType column in this table is used to identify employees as executive, administrative, or consultants. You need to ensure that the administrative employees can add, update, or delete data for non-executive employees only.

What should you do?

A. Create a view, and include the WITH ENCRYPTION clause.

B. Create a view, and include the WITH CHECK OPTION clause.

C. Create a view, and include the SCHEMABINDING clause.

D. Create a view, and build a covering index on the view.

E. Create a user-defined function that returns a table containing the non-executive employees.

Answer: B.
Explanation: By default, as rows are added or updated through a view, they disappear from the scope of the view when they no longer fall into the criteria of the query defining the view. The WITH CHECK OPTION clause forces all data modification statements that are executed against a view to adhere to the criteria set within the SELECT statement. When a row is modified through the view, the WITH CHECK OPTION ensures that the rows cannot be modified in a way that causes them to disappear from the view. Any modification that would cause this to happen is cancelled and an error message is displayed. By using the WITH CHECK OPTION and setting the view criteria to specify that the rows pertaining to the EmpType columns that contain the value "executive" cannot be altered, administrative employees will be prevented from modifying the data in those rows.

Incorrect answers:

A: The WITH ENCRYPTION clause cannot be used to ensure that only specific rows of a view can be modified.

Note: The WITH ENCRYPTION clause causes SQL Server to encrypt the system table columns containing the text of the CREATE VIEW statement. Using the WITH ENCRYPTION clause prevents the users from acquiring the definition (or code) of the view.

C: The SCHEMABINDING clause cannot be used to ensure that only specific rows of a view can be modified.

Note: The SCHEMABINDING clause binds the view to the schema. When SCHEMABINDING is specified, the SELECT statement must include the two-part names of tables, views, or user-defined functions that is referenced. Views or tables participating in a view created with the SCHEMABINDING clause cannot be dropped unless that view is dropped or changed so that it no longer has schema binding. Furthermore, ALTER TABLE statements on tables that participate in views having SCHEMABINDING will fail if these statements affect the view definition.

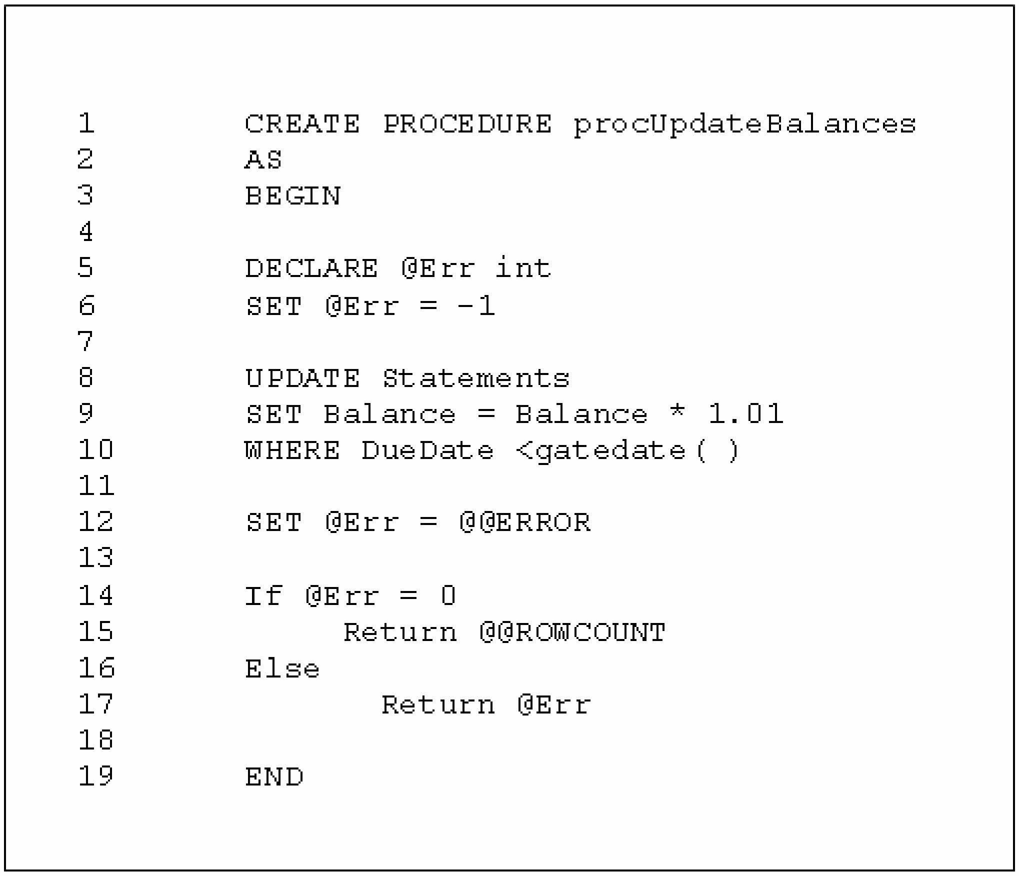
D: A COVERING INDEX cannot be used to ensure that only specific rows of a view can be modified.

Note: A COVERING INDEX is an index that contains all the referenced columns and is one of the fastest ways to access data. Because the data pages do not have to be accessed at all, physical disk I/O is reduced.

E: The function only returns a result set. The administrative employees would not be able to modify these rows.

Note: Functions are similar to stored procedure. They are subroutines made up of one or more Transact-SQL statements that can be used to encapsulate code for reuse. SQL Server 2000 does not limit users to the built-in functions defined as part of the Transact-SQL language, but allows users to create their own user-defined functions. Each user-defined function must have a unique name, and the creator of the user-defined function must be granted CREATE FUNCTION permissions to create, alter, or drop user-defined functions. Users other than the owner must be granted appropriate permissions on a function before they can use it in a Transact-SQL statement. In order to create or alter tables through the use of user-defined functions that reference the CHECK constraint, DEFAULT clause, or computed column definition, users must also have REFERENCES permission on those functions.

Q. 46
You are a database developer for an electric utility company. When customers fail to pay the balance on a billing statement before the statement due date, the balance of the billing statement needs to be increased by 1 percent each day until the balance is paid. The company needs to track the number of overdue billing statements. You create a stored procedure to update the balances and to report the number of billing statements that are overdue. The stored procedure is shown in the exhibit.



```
1       CREATE PROCEDURE procUpdateBalances
2       AS
3       BEGIN
4
5       DECLARE @Err int
6       SET @Err = -1
7
8       UPDATE Statements
9       SET Balance = Balance * 1.01
10      WHERE DueDate <gatedate( )
11
12      SET @Err = @@ERROR
13
14      If @Err = 0
15           Return @@ROWCOUNT
16      Else
17             Return @Err
18
19      END
```

Each time the stored procedure executes without error, it reports that zero billing statements are overdue. However, you observe that balances are being updated by the procedure.

What should you do to correct the problem?

A. Replace the lines 12-17 of the stored procedure with the following:

```
Return @@ROWCOUNT
```

B. Replace line 5-6 of the stored procedure with the following:

```
DECLARE @count int
```

Replace lines 12-17 with the following:

```
SET @Count = @@ROWCOUNT
If @@ERROR = 0
     Return @Count
Else
     Return -1
```

C. Replace line 5 of the stored procedure with the following:
```
DECLARE @Err int, @Count int
```

Replace lines 12-17 with the following:
```
SELECT @Err = @@ERROR, @Count = @@ROWCOUNT
IF @Err = 0
     Return @Count
Else
     Return @Err
```

D. Replace line 5 of the stored procedure with the following:
```
Return @@Error
```

E. Replace line 5 of the stored procedure with the following
```
DECLARE @Err int, @Count int
```

Replace line 9 with the following:
```
SET Balance = Balance * 1.01, @Count = Count (*)
Replace line 15 with the following:
Return @Count
```

Answer: C.
Explanation: This solution works; see details below. It would return either the error code or the number of rows affected. This seems to be the intention of the original code of the stored procedure.

Discussion of solution:
The @@ROWCOUNT global variable (or function – you could have different opinions on what it really is) returns the number of rows affected by the last statement. The @@ROWCOUNT is reset for every new statement that executes. And that is why the original procedure incorrectly reports that zero billing statements are overdue.

A statement can be split up to several lines to improve readability. In this scenario, the last statement before the Return @@ROWCOUNT statement would be:

8. UPDATE Statements
9. SET Balance = Balance * 1.01
10. WHERE DueDate <gatedate( )

We must make sure to catch the @@ROWCOUNT variable here, or else it would become reset.
First we declare a local @Count variable by replacing line 5 with:
DECLARE @Err int, @Count int

Then we catch the @ROWCOUNT and the @ERROR variables into the local variable with:
SELECT @Err = @@ERROR, @Count = @@ROWCOUNT

Finally the error handling is implemented with:

```
IF @Err = 0
        Return @Count
Else
        Return @Err
```

If no error occurred then it returns the numbers of row affected. If an error occurred, the number of the error is returned.

Note: When a statement is executed the @@ERROR is set to 0 if the statement executed successfully. If an error occurs, an error message is returned. @@ERROR returns the number of the error message until another Transact-SQL statement is executed.

Incorrect answers:

A: This simplistic solution would return the number of affected rows, but it would not return any indication of errors. The original solution, however, indicates that procedure is intended to return either the number of rows or the error number.

B: A local variable @COUNT is declared, and then it is suppose to catch the @@ROWCOUNT variable but that statement is incorrect:

SET @Count = @ROWCOUNT
The @ROWCOUNT is a reference to an undeclared local variable, not the function @@ROWCOUNT.

There is another problem with this code as well. Even if it were changed it would not work. The @@ROWCOUNT and the @@ERROR values must be stored in the same statement. The @@ERROR = 0 statement just checks whether the statement SET @Count = @@ROWCOUNT executed correctly.

```
SET @Count = @@ROWCOUNT
If @@ERROR = 0
```

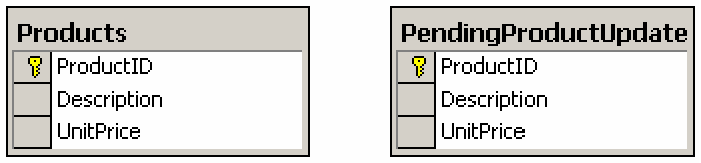
D: The @@ROWCOUNT, not the @@ERROR, should be returned.

E: The following statement is incorrect:

```
SET Balance = Balance * 1.01, @Count = Count (*)
```

The @Count = Count (*) doesn't work. The @Count variable must catch the value of the @@ROWCOUNT function.

Q. 47
You are a database developer for an online electronics company. The company's product catalog is contained in a table named Products. The Products table is frequently accessed during normal business hours. Modifications to the Products table are written to a table named PendingProductUpdate. These tables are shown in the exhibit.



The PendingProductUpdate table will be used to update the Products table after business hours. The database server runs SQL Server 2000 and is set to 8.0 compatibility mode.

You need to create a script that will be used to update the products table. Which script should you use?

A.
```
UPDATE Products
SET p1.[Description] = p2.[Description], p1.UnitPrice = p2.UnitPrice
FROM Products p1, PendingProductUpdate p2
WHERE p1.ProductID= p2.ProductID
GO
TRUNCATE TABLE PendingProductUpdate
GO
```

B.
```
UPDATE Products p1
SET [Description] = p2.[Description], UnitPrice = p2.UnitPrice
FROM Products, PendingProductUpdate p2
WHERE p1.ProductID= p2.ProductID
GO
TRUNCATE TABLE PendingProductUpdate
GO
```

C.
```
UPDATE Products p1
SET p1.[Description] = p2.[Description], p1.UnitPrice = p2.UnitPrice
FROM (SELECT [Description], UnitPrice
        FROM PendingProductUpdate p2
        WHERE p1.ProductID= p2.ProductID)
```

```
GO
TRUNCATE TABLE PendingProductUpdate
GO
```

D.
```
UPDATE p1
SET p1.[Description] = p2.[Description], p1.UnitPrice = p2.UnitPrice
FROM Products p1, PendingProductUpdate p2
WHERE p1.ProductID= p2.ProductID
GO
TRUNCATE TABLE PendingProductUpdate
```

Answer: D.
Explanation: This code is correct. Both aliases p1 and p2 are defined in the FROM statement.

Incorrect answers:

A: The alias p1 defined in the FROM statement must be used in the UPDATE statement in order to be used in the left-hand side of the '=' sign.

B: This code will not run. An alias cannot be defined in the UPDATE statement and then used in the WHERE statement. The p1 alias is defined and used in an incorrect way.

C: The FROM clause is incorrect. There should be no SELECT statement in the FROM clause; instead there should be table names.

Q. 48
You are the database developer for a sporting goods company that exports products to customers worldwide. The company stores its sales information in a database named Sales. Customer names are stored in a table named Customers in this database. The script that was used to create this table is shown in the exhibit.

```
CREATE TABLE Customers
     (
     CustomerID int NOT NULL,
     CustomerName varchar(30) NOT NULL,
     ContactName varchar(30) NULL,
     Phone varchar(20) NULL,
     Country varchar(30) NOT NULL,
     CONSTRAINT PK_Customers PRIMARY KEY (CustomerID)
     )
```

There are usually only one or two customers per country. However, some countries have as many as 20 customers. Your company's marketing department wants to target its advertising to countries that have more than 10 customers.

You need to create a list of these countries for the marketing department. Which script should you use?

A.
```
SELECT Country FROM Customers
GROUP BY Country HAVING COUNT (Country)>10
```

B.
```
SELECT TOP 10 Country FROM Customers
```

C.
```
SELECT TOP 10 Country FROM Customers
FROM (SELECT DISTINCT Country FROM Customers) AS X
GROUP BY Country HAVING COUNT(*)> 10
```

D.
```
SET ROWCOUNT 10
SELECT Country, COUNT (*) as "NumCountries"
FROM Customers
GROUP BY Country ORDER BY NumCountries, Desc
```

Answer: A.
Explanation: The HAVING clause specifies a search condition for a group or an aggregate. HAVING can be used only with the SELECT statement. It is usually used in a GROUP BY clause. After the data has been grouped and aggregated, the conditions in the HAVING clause are applied and only the groups that meet the conditions appear in the query. When GROUP BY is not used, HAVING behaves like a WHERE clause.

Incorrect answers:

B: The TOP 10 option of the SELECT statement is used to select 10 rows, and it should always be used in conjunction with an ORDER BY clause. In this scenario, we cannot restrict the number of rows with the TOP option; we must use a WHERE or HAVING statement instead since we want to put a restriction on one of the columns.

C: This code is incorrect as it contains two successive FROM clauses. A UNION clause should be used to combine the results sets from two FROM clauses.

D: This code would give a result set with the 10 countries that have the most customers. However, we are interested in all countries that more than 10 customers.

Note: The SET ROWCOUNT clause was used prior to SQL Server 7 to limit SQL Server to sending only a specified number of rows to the client. Every SELECT statement would stop sending rows back to the client after the specified number of rows had been sent. This SET ROWCOUNT clause is the functionally equivalent to the SELECT TOP clause, hence SET ROWCOUNT 10 performs the same function as SELECT TOP 10. Furthermore, SET ROWCOUNT overrides the SELECT statement TOP keyword if the rowcount is the smaller value. The use of the SELECT TOP clause is recommended, as SQL 2000 processes it more efficiently than the SET ROWCOUNT clause.

Q. 49
You are a database developer for a sales organization. Your database has a table named Sales that contains summary information regarding the sales orders from salespeople. The sales manager asks you to create a report of the salespeople who had the 20 highest total sales.

Which query should you use to accomplish this?

A.
```
SELECT TOP 20 PERCENT LastName, FirstName, SUM (OrderAmount) AS ytd
FROM sales
GROUP BY LastName, FirstName
ORDER BY 3 DESC
```

B.
```
SELECT LastName, FirstName, COUNT(*) AS sales
FROM sales
GROUP BY LastName, FirstName
HAVING COUNT (*) > 20
ORDER BY 3 DESC
```

C.
```
SELECT TOP 20 LastName, FirstName, MAX(OrderAmount) AS ytd
FROM sales
GROUP BY LastName, FirstName
ORDER BY 3 DESC
```

D.
```
SELECT TOP 20 LastName, FirstName, SUM (OrderAmount) AS ytd
FROM sales
GROUP BY LastName, FirstName
ORDER BY 3 DESC
```

E.
```
SELECT TOP 20 WITH TIES LastName, FirstName, SUM (OrderAmount) AS ytd
FROM sales
GROUP BY LastName, FirstName
ORDER BY 3 DESC
```

Answer: E.
Explanation: As we want a report containing the salespeople that have the 20 highest total sales, we must use an SQL query which has a FROM clause that contains TOP 20 WITH TIES. We must also make sure that the result set is ordered on the Total Sales column, the third column, in descending order. This is accomplished with ORDER BY 3 DESC.

Note 1: The SELECT TOP clause limits the number of rows returned in the result set and specifies how many rows must be returned either as an absolute number or as a percentage. SELECT TOP 20 specifies that only the top 20 rows should be output from the query result set. Furthermore, the WITH TIES clause used in conjunction with the SELECT TOP statement specifies that additional rows be returned from the base result set that have same value in the ORDER BY columns as the last of the rows returned. SELECT TOP ...WITH TIES can only be specified when an ORDER BY clause is specified.

Note 2: Aggregate functions such as SUM, AVG, COUNT, COUNT(*), MAX, and MIN generate summary values in a query result set. With the exception of COUNT(*), an aggregate function processes all the selected values, ignoring null values, in a single column to generate a single result value. Aggregate functions can be applied to all rows in a table, to a subset of the table specified by a WHERE clause, or to one or more groups of rows in the table. The keyword DISTINCT can be used with SUM, AVG, and COUNT to eliminate duplicate values before an aggregate function is applied. SUM and AVG can be used only with numeric columns, MIN and MAX cannot be used with bit data types and aggregate functions other than COUNT(*) cannot be used with text and image data types. Aggregate functions cannot be used in a WHERE clause. However, a SELECT statement with aggregate functions in its select list often includes a WHERE clause that restricts the rows to which the aggregate function is applied. If a SELECT statement includes a WHERE clause (but not a GROUP BY clause), an aggregate function produces a single value for the subset of rows specified by the WHERE clause.

Incorrect answers:

A: The SELECT TOP clause limits the number of rows returned in the result set and specifies how many rows must be returned either as an absolute number or as a percentage. SELECT TOP 20 PERCENT specifies that only the top 20% of rows in the query result set should be output. However, the scenario requires that number of rows output should be an absolute number.

B: COUNT (*) specifies that all rows should be counted to return the total number of rows in a table without eliminating duplicates. It cannot be used with DISTINCT and does not require an expression parameter because it does not use information about any particular column. This aggregate function does not return the values of particular columns and is thus not appropriate to this scenario.

C: MAX returns the highest value in a column. It returns only one row and thus does not meet the requirements of this scenario.

D: This code does meet the requirements of the scenario; however, it does not return more than 20 rows Thus, if there is a tie for 20th place, only the first 20 rows will be returned and not the succeeding rows that have the same aggregate value.

Q. 50
You are a database developer for a travel agency. A table named FlightTimes in the Airlines database contains flight information for all airlines. The travel agency uses an intranet-based application to manage travel reservations. This application retrieves flight information for each airline from the FlightTimes table. Your company primarily works with one particular airline. In the Airlines database, the unique identifier for this airline is 101.

The application must be able to request flight times without having to specify a value for the airline. The application should be required to specify a value for the airline only if a different airline's flight times are needed.

What should you do?

A. Create two stored procedures, and specify that one of the stored procedures should accept a parameter and that the other should not.

B. Create a user-defined function that accepts a parameter with a default value of 101.

C. Create a stored procedure that accepts a parameter with a default value of 101.

D. Create a view that filters the **FlightTimes** table on a value of 101.

E. Create a default of 101 on the **FlightTimes** table.

Answer: C.
Explanation: Stored procedures with optional parameters can be created by specifying a default value for optional parameters. When the stored procedure is executed, the default value is used if no other value has been specified. Specifying default values is necessary because a system error is returned if a parameter does not have a default value specified in the stored procedure and the calling program does not provide a value for the parameter when the stored procedure is executed.

Incorrect answers:
A: Two procedures would not, by themselves, solve the problem in this scenario. Two procedures are not required; the problem can be solved with one procedure that uses a parameter that has a default value.

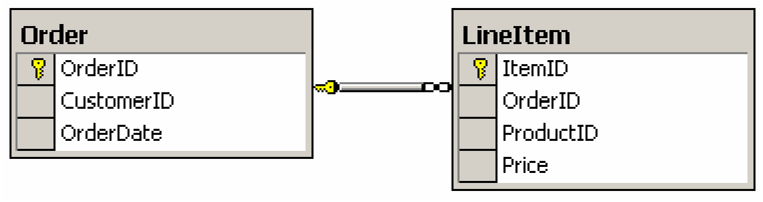
B: Functions are subroutines made up of one or more Transact-SQL statements that can be used to encapsulate code for reuse. SQL Server 2000 does not limit users to the built-in functions but allows users to create their own user-defined functions through the use of the CREATE FUNCTION statement.

A user-defined function takes zero or more input parameters. When a parameter of the function has a default value, the keyword DEFAULT must be specified when calling the function to get the default value. This behavior is different from parameters with default values in stored procedures in which omitting the parameter also implies the default value. User-defined functions do not support output parameters.

D: This approach is incorrect. Table views are used to restrict access to the base table and must specify which columns or rows to return. A view can be created that will only return the row for which the unique identifier value is 101. This, however, will return one row only and the application will thus only have access to that row.

E: Default values for unique identifiers cannot be set on the table.

Q. 51
You are a database developer for Wingtip Toys. You have created an order entry database that includes two tables, as shown in the exhibit.



Users enter orders into an entry application. When a new order is entered, the data is saved to the Order and LineItem tables in the order entry database.

You must ensure that the entire order is saved successfully. Which script should you use?

A.
```
BEGIN TRANSACTION Order
INSERT INTO Order VALUES (@ID, @CustomerID, @OrderDate)
INSERT INTO LineItem VALUES (@ItemID, @ID, @ProductID, @Price)
SAVE TRANSACTION Order
```

B.
```
INSERT INTO Order VALUES (@ID, @CustomerID, @OrderDate)
INSERT INTO LineItem VALUES (@ItemID, @ID, @ProductID, @Price)

IF (@@Error = 0)
        COMMIT TRANSACTION
ELSE
        ROLLBACK TRANSACTION
```

C.
```
BEGIN TRANSACTION
INSERT INTO Order VALUES (@ID, @CustomerID, @OrderDate)
IF (@@Error = 0)
BEGIN
    INSERT INTO LineItem
    VALUES (@ItemID, @ID, @ProductID, @Price)

        IF (@@Error = 0)
            COMMIT TRANSACTION
        ELSE
            ROLLBACK TRANSACTION
END
ELSE
    ROLLBACK TRANSACTION
END
```

D.
```
BEGIN TRANSACTION
INSERT INTO Order VALUES (@ID, @CustomerID, @OrderDate)
IF (@@Error = 0)
    COMMIT TRANSACTION
ELSE
    ROLLBACK TRANSACTION
BEGIN TRANSACTION
    INSERT INTO LineItem VALUES (@ItemID, @ID, @ProductID, @Price)
IF (@@Error = 0)
    COMMIT TRANSACTION
ELSE
    ROLLBACK TRANSACTION
```

Answer: C.
Explanation: This code contains a single transaction. This transaction includes insertion of a row into the Order table and the insertion of a row into the LineItem table. Both this insertions must succeed in order for the transaction to commit. This meets the requirement of this scenario.

Incorrect answers:

A: The SAVE TRANSACTION function in line 4 of this code sets a savepoint within the transaction. A savepoint defines a location to which a transaction can return if part of the transaction is conditionally cancelled. If a transaction is rolled back to a savepoint, it must proceed to completion with more Transact-SQL statements if needed and a COMMIT TRANSACTION statement, or it must be cancelled altogether using ROLLBACK TRANSACTION. This code has neither COMMIT TRANSACTION nor ROLLBACK TRANSACTION.

B: This code does not have a BEGIN TRANSCATION command and hence does not contain a defined transaction. Consequently, it will produce an error once one of the conditions is met.

D: This code contains two transactions, each with its own BEGIN TRANSACTION, its own IF condition and its own COMMIT TRANSACTION. These two transactions follow each other. Thus when the IF condition in the first transaction is met, the transaction will be committed regardless of the outcome of the IF condition in the second TRANSACTION. Hence, this code does not meet the requirements set out in the scenario.

Q. 52
You are a database developer for a vacuum sales company. The company has a database named Sales that contains tables named VacuumSales and Employee. Sales information is stored in the VacuumSales table. Employee information is stored in the Employee table. The Employee table has a bit column named IsActive. This column indicates whether an employee is currently employed. The Employee table also has a column named EmployeeID that uniquely identifies each employee. All sales entered into the VacuumSales table must contain an employee ID of a currently employed employee.

How should you enforce this requirement?

A. Use the Microsoft Distributed Transaction Coordinator to enlist the **Employee** table in a distributed transaction that will roll back the entire transaction if the employee ID is not active.

B. Add a CHECK constraint on the **EmployeeID** column of the **VacuumSales** table.

C. Add a FOREIGN KEY constraint on the **EmployeeID** column of the **VacuumSales** table that references the **EmployeeID** column in the **Employee** table.

D. Add a FOR INSERT trigger on the **VacuumSales** table. In the trigger, join the **Employee** table with the **inserted** table based on the **EmployeeID** column, and test the **IsActive** column.

Answer: D.
Explanation: Triggers can perform the same function as CHECK constraints and FOREIGN KEY constraints. Where possible, CHECK constraints or FOREIGN KEY constraints should be used. In this scenario neither CHECK constraints, which ensure that valid data is entered into a column, nor FOREIGN KEY constraints, which ensure referential integrity, can meet the functional requirements set by the scenario. That is, they would not be able to check the value of the IsActive column of the Employee table when inserting rows into the VacuumSales table. Triggers, on the other hand, can use other tables when they execute.

Incorrect answers:

A: Distributed transactions allows users to create transactions that update multiple SQL Server databases and other sources of data, as distributed transactions involve resources from two or more sources. Distributed transactions are not suitable for this scenario.

B: Check constraints can only be used within a table. Check constraints cannot be used to compare rows in two different tables.

Note: CHECK constraints are used to enforce domain integrity by checking that no values designated as unacceptable for a column are placed in that column. They are similar to FOREIGN KEY constraints in that they control the values that are placed in a column. FOREIGN KEY constraints get the list of valid values from another table, while CHECK constraints determine the valid values from a logical expression that is not based on data in another column or table.

C: Foreign key constraints can only be used to compare equality of two different columns, the foreign key column and the unique column, in different tables. Foreign key constraints cannot be used to test values of other columns.

Note on FOREIGN KEY constraints: The constraint enforces referential integrity by ensuring that changes cannot be made to data in the primary key table if those changes invalidate the link to data in the foreign key table. If an attempt is made to delete a row in a primary key table or to change a primary key value, the action will fail if the deleted or changed primary key value corresponds to a value in the FOREIGN KEY constraint of another table.

Q. 53

You are a database developer for an online brokerage firm. The prices of the stocks owned by customers are maintained in a SQL Server 2000 database.

To allow tracking of the stock price history, all updates of stock prices must be logged. To help correct problems regarding price updates, any errors that occur during an update must also be logged. When errors are logged, a message that identifies the stock producing the error must be returned to the client application.

You must ensure that the appropriate conditions are logged and that the appropriate messages are generated. Which procedure should you use?

A.
```
CREATE PROCEDURE UpdateStockPrice @StockID int, @Price decimal
AS BEGIN

DECLARE @Msg varchar(50)
```

```
UPDATE Stocks SET CurrentPrice = @Price
WHERE StockID = @ StockID
AND CurrentPrice <> @ Price

IF @@ERROR <> 0
    RAISERROR (‘Error %d occurred updating Stock %d.’, 10, 1, @@ERROR, @StockID) WITH
LOG
IF @@ROWCOUNT > 0
BEGIN
    SELECT @Msg = ‘Stock’ + STR (@StockID) + ‘updated to’ + STR (@Price) + ‘.’
    EXEC master. . xp_LOGEVENT 50001, @Msg
END

END
```

B.

```
CREATE PROCEDURE UpdateStockPrice @StockID int, @Price decimal
AS BEGIN

UPDATE Stocks SET CurrentPrice = @Price
WHERE StockID = @ StockID
AND CurrentPrice <> @ Price

IF @@ERROR <> 0
    PRINT ‘ERROR’ + STR(@@ERROR) + ‘occurred updating Stock’ +STR (@StockID)+ ‘.’
IF @@ROWCOUNT > 0
    PRINT ‘Stock’ + STR (@StockID) + ‘updated to’ + STR (@Price) + ‘.’

END
```

C.

```
CREATE PROCEDURE UpdateStockPrice @StockID int, @Price decimal
AS BEGIN

DECLARE @Err int, @RCount int, @Msg varchar(50)

UPDATE Stocks SET CurrentPrice = @Price
WHERE StockID = @ StockID
AND CurrentPrice <> @ Price

SELECT @Err = @@ERROR, @RCount = @@ROWCOUNT
IF @Err <> 0
BEGIN
    SELECT @Msg = ‘Error’ + STR(@Err) + ‘occurred updating Stock’ + STR (@StockID) + ‘.’
    EXEC master..xp_logevent 50001, @Msg
```

```
END
IF @RCOUNT > 0
BEGIN
     SELECT @Msg = ‘Stock’ + STR (@StockID) + ‘updated to’ + STR (@Price) + ‘.’
     EXEC master. . xp_LOGEVENT 50001, @Msg
END

END
```

D. CREATE PROCEDURE UpdateStockPrice @StockID int, @Price decimal AS BEGIN

```
DECLARE @Err int, @RCount int, @Msg varchar (50)

UPDATE Stocks SET CurrentPrice = @Price
WHERE StockID = @StockID
AND CurrentPrice <> @Price

SELECT @Err = @@ERROR, @RCount = @@ROWCOUNT
If @Err <> 0
     RAISEERROR (‘Error %d occurred updating Stock %d.’, 10, 1, @Err, @StockID) WITH LOG
If @RCount > 0
BEGIN
     SELECT @Msg = ‘Stock’ + STR (@StockID) + ‘update to’ + STR (@Price) + ‘.’
     EXEC master. . xp_logevent 50001, @Msg
END

END
```

Answer: D.
Explanation:
The following three requirements must be met:

Á All updates of stock prices must be logged.
Á Any error that occurs during a price update must be logged.
Á When errors are logged, a message that identifies the stock producing the error must be returned to the client application.

In the event of an error, you might want to display a message describing the reason for the error. The RAISERROR system command is used to display error messages. To log the error the WITH LOG option has to be used. The WITH LOG option logs the error in the server error log and the application log.

The xp_logevent extended stored procedure logs a user-defined message in the Microsoft SQL Server log file and in the Microsoft Windows 2000/NT Event Viewer. xp_logevent can be used to send an alert without sending a message to the client.

When sending messages from Transact-SQL procedures, triggers, batches, and so on, use the RAISERROR statement instead of xp_logevent. xp_logevent does not call a client's message handler or set @@ERROR. To write messages to the Windows NT Event Viewer and to the SQL Server error log file within SQL Server, execute the RAISERROR statement.

It is also important to save the @@ROWCOUNT variable into a local variable with the following statement:

```
SELECT @Err = @@ERROR, @RCount = @@ROWCOUNT
```

The @@ROWCOUNT variable is set to 0 by any statement that does not return rows, such as an IF statement.

Incorrect answers:

A: This code looks pretty good. There is just one small problem. At the time when the statement **IF @@ROWCOUNT > 0** is executed, the @@ROWCOUNT variable has already been reset by the previous IF-statement. The @@ROWCOUNT variable is set to 0 by any statement that does not return rows, such as an IF statement.

B: Errors must be logged, so the RAISEERROR statement must be used. Just printing the error with a PRINT command will not log the error.

C: Errors must be shown. Here they only will be logged with statement,

```
EXEC master..xp_logevent 50001, @Msg
```

Q. 54

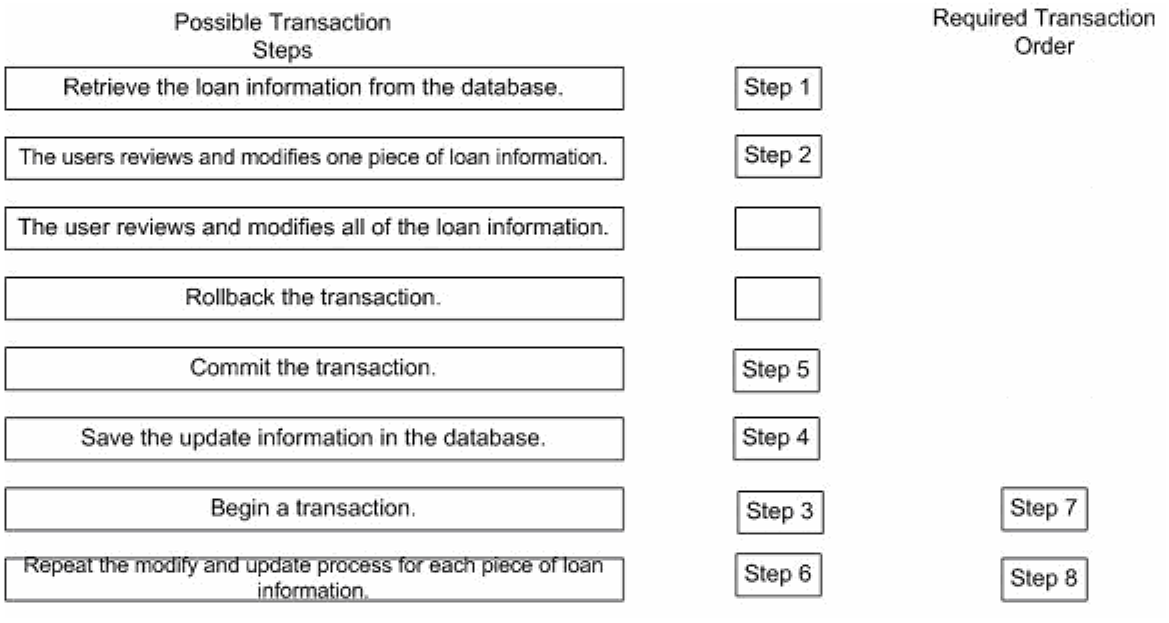
You are a database developer for a loan servicing company. You are designing database transactions to support a new data entry application. Users of the new data entry application will retrieve loan information from a database. Users will make any necessary changes to the information and save the updated information to the database.

How should you design these transactions?

To answer, click the Select and Place button, and then drag the appropriate transaction order choices beside the appropriate transaction steps. (Use only order choices that apply)

Possible Transaction Steps		Required Transaction Order
Retrieve the loan information from the database.		Step 1
The users reviews and modifies one piece of loan information.		Step 2
The user reviews and modifies all of the loan information.		Step 3
Rollback the transaction.		Step 4
Commit the transaction.		Step 5
Save the update information in the database.		Step 6
Begin a transaction.		Step 7
Repeat the modify and update process for each piece of loan information.		Step 8

Answer:


Possible Transaction Steps
Required Transaction Order
Retrieve the loan information from the database.
Step 1
The users reviews and modifies one piece of loan information.
Step 2
The user reviews and modifies all of the loan information.
Rollback the transaction.
Commit the transaction.
Step 5
Save the update information in the database.
Step 4
Begin a transaction.
Step 3
Step 7
Repeat the modify and update process for each piece of loan information.
Step 6
Step 8

Explanation:

Step 1: Retrieve loan information from the database.
Step 2: The user reviews and modifies a piece of loan information
Step 3: Begin a transaction
Step 4: Save the update information in the database.
Step 5: Commit the transaction
Step 6: Repeat the modify and update process for each piece of loan information

General notes on transactions:

Á Keep short executive actions in a transaction.

Á Never leave pending user input action in your transaction loop. A transaction cannot be kept waiting for input from a user.

Taking these considerations in mind, the logic of the reasoning is as follows:

Á First we retrieve loan information from the database to the local application. Then the user reviews and modifies only one piece of loan information, not all the loan information, since we want to keep the amount of information updated in a transaction as small as possible.

Á Then we start the transaction. We need to start the transaction to make sure no one else updates the same data. The information needs to be saved in the database, since it's only modified locally at the client. After the data is updated, the transaction is committed. We don't have to put in any steps for rollback the transaction. The transaction will automatically be rolled back if the update fails.

Á This whole procedure is repeated for each piece of loan information.

Q. 55
You are a database developer for a company that leases trucks. The company has created a web site that customer can use to reserve trucks. You are designing the SQL server 2000 database to support the web site.

New truck reservations are inserted into a table named Reservations. Customers who have reserved a truck can return to the web site and update their reservation. When a reservation is updated, the entire existing reservation must be copied to a table named History.

Occasionally, customers will save an existing reservation without actually changing any of the information about the reservation. In this case, the existing reservation should not be copied to the History table.

You need to develop a way to create the appropriate entries in the History table.

What should you do?

A. Create a trigger on the **Reservations** table to create the **History** table entries.

B. Create a cascading referential integrity constraint on the **Reservations** table to create the **History** table entries.

C. Create a view on the **Reservations** table. Include the WITH SCHEMABINDING option in the view definition.

D. Create a view on the **Reservations** table. Include the WITH CHECK OPTION clause in the view definition.

Answer: A.
Explanation: Triggers are a special class of stored procedure that execute automatically when an UPDATE, INSERT, or DELETE statement is issued against a table or view. Triggers are powerful tools that can be used to enforce business rules automatically when data is modified. Triggers can also extend the integrity checking logic of SQL Server constraints, defaults, and rules, although constraints and defaults should be used instead whenever they provide all the needed functionality. In this scenario, rows must be copied to the History table when an UPDATE occurs on the Reservation table. When an existing reservation is saved, an UPDATE does not occur, as there is no data modification. Thus when customers save their existing reservation, the trigger does not fire.

Incorrect answers:

B: Cascading referential integrity cannot be used to copy a row from one table to another.

Note: Cascading referential integrity constraints are used to ensure the referential integrity of data through the use of FOREIGN KEYS. Cascading referential actions fire AFTER triggers ensuring that the columns that hold FOREIGN KEY constraints with the column that is being modified are also modified so that they hold the same data.

C: A view in general, or more specifically, a view with the schema binding option, cannot be used to copy rows from one table to another when one row changes in one of the tables.

Note: When a view is created with the SCHEMABINDING option, the view is bound to the schema. Views or tables participating in a view created with the schema binding clause cannot be dropped unless that view is dropped or changed so it no longer has schema binding. Otherwise, SQL Server raises an error. Furthermore, ALTER TABLE statements on tables that participate in views having schema binding will fail if these statements affect the view definition.

D: A view in general, or more specifically, a view with the WITH CHECK OPTION, cannot be used to copy rows from one table to another when one row changes in one of the tables.

Note: As rows change when they are updated through a view, they could fall outside of the scope of the view if they no longer fall into the criteria of the query that defined the view. When this occurs the affected rows could disappear from the view. The WITH CHECK OPTION clause forces all data modification statements executed against the view to adhere to the criteria set within the SELECT statement defining the view. The use of this clause ensures that rows cannot be modified in a way that causes them to disappear from the view. Any modification that would cause this is cancelled and an error is displayed.

Q. 56
You are a database developer for Proseware, Inc. The company has a database that contains information about companies located within specific postal codes. This information is contained in the company table within this database. Currently, the database contains company data for five different postal codes. The number of companies in a specific postal code currently ranges from 10 to 5,000. More companies and postal codes will be added to the database over time.

You are creating a query to retrieve information from the database. You need to accommodate new data by making only minimal changes to the database. The performance of your query must not be affected by the number of companies returned.

You want to create a query that performs consistently and minimizes future maintenance. What should you do?

A. Create a stored procedure that requires a postal code as a parameter. Include the WITH RECOMPILE option when the procedure is created.

B. Create one stored procedure for each postal code.

C. Create one view for each postal code.

D. Split the **Company** table into multiple tables so that each table contains one postal code. Build a partitioned view on the tables so that the data can still be viewed as a single table.

Answer: A.
Explanation: We need to meet the following requirements:

1. The new data, more companies and postal codes, must be added with minimal changes to the database.
2. Future maintenance should be minimized.
3. The performance of the query should not be affected by the number of companies returned.
4. The query should perform consistently.

A single procedure with a postal code parameter meets the first requirement. We only add a single procedure and make no further changes to the existing database.

The second requirement is met since no adjustments have to be made for additional companies and postal codes. The WITH RECOMPILE option will force a recompilation of the execution plan every time the stored procedure is run. This could decrease the performance, especially if the stored procedure is used frequently.

The third requirement is met because the performance of the query is not affected by the companies returned, since the execution is recompiled every time the stored procedure is run.

The fourth requirement, a consistent performance of the query, is also met by the same argument.

Note: Specifying the WITH RECOMPILE option in stored procedure definition indicates that SQL Server should not cache a plan for this stored procedure; the stored procedure is recompiled each time it is executed. The use of the WITH RECOMPILE option causes the stored procedure to execute more slowly because the stored procedure must be recompiled each time it is executed. This option should only be used when stored procedures take parameters whose values differ widely between executions of the stored procedure, resulting in different execution plans to be created each time

Reference:
INF: Troubleshooting Stored Procedure Recompilation (Q243586)
BOL, Creating a Partitioned View
BOL, Using Partitioned Views
SQL Server 2000 Resource Kit, Chapter 38 - Scaling Out on SQL Server

Incorrect answers:

B: Creating one stored procedure for each postal code would not minimize future maintenance. A new procedure would have to be created for every new postal code that is added. Furthermore, it would not, like a portioned view, have any performance gains.

C: Creating one view for each postal code would not minimize future maintenance. A new view would have to be created for every new postal code that is added. Furthermore, it would not, like a portioned view, have any performance gains.

D: Using a portioned view to split the table into separate tables for each postal code might improve performance somewhat, though partitioning data on the same server does not ordinarily provide noticeable performance gains. The 3rd and 4th requirements are met.

It would not, however, meet the 1st and 2nd requirements. Initially we would have to create five new tables, one for each postal code, and one portioned view. We would then constantly have to add a new table and adjust the partitioned view for every new postal code being used in the database.

A portioned view would be the best solution if the requirements of the scenario were for improved performance and not minimized maintenance.

Q. 57
You are a database developer for Woodgrove Bank. You are implementing a process that loads data into a SQL Server 2000 database. As a part of this process, data is temporarily loaded into a table named Staging. When the data load process is complete, the data is deleted from this table. You will never need to recover this deleted data.

You need to ensure that the data from the Staging table is deleted as quickly as possible. What should you do?

A. Use a DELETE statement to remove the data from the table.

B. Use a TRUNCATE TABLE statement to remove the data from the table.

C. Use a DROP TABLE statement to remove the data from the table.

D. Use an updateable cursor to access and remove each row of data from the table.

Answer: B.
Explanation: If you want to delete all the rows in a table, TRUNCATE TABLE is faster than DELETE. DELETE physically removes rows one at a time and records each deleted row in the transaction log. TRUNCATE TABLE deallocates all pages associated with the table. For this reason, TRUNCATE TABLE is faster and requires less transaction log space than DELETE. TRUNCATE TABLE is functionally equivalent to DELETE with no WHERE clause, but TRUNCATE TABLE cannot be used with tables referenced by foreign keys. Both DELETE and TRUNCATE TABLE make the space occupied by the deleted rows available for the storage of new data

Incorrect answers:
A: The DELETE statement may fail if it violates a trigger or attempts to remove a row referenced by data in another table with a FOREIGN KEY constraint. If the DELETE removes multiple rows, and any one of the removed rows violates a trigger or constraint, the statement is cancelled, an error is returned, and no rows are removed.

C: We just want to delete all rows in the table; we don't want to delete the table itself. The DROP TABLE removes a table definition and all data, indexes, triggers, constraints, and permission specifications for that table. Any view or stored procedure that references the dropped table must be explicitly dropped by using the DROP VIEW or DROP PROCEDURE statement.

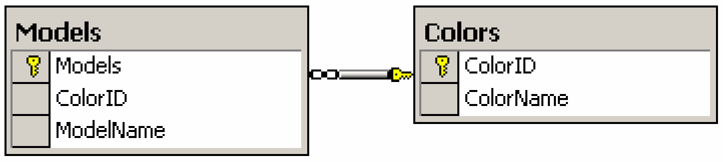
D: A cursor that deletes one row at a time can be used to delete all rows in a table. However, TRUNCATE TABLE is faster and uses fewer system and transaction log resources than multiple DELETE statements.

Note: In a relational database, operations act on a complete set of rows that satisfy the conditions in the WHERE clause of the SELECT statement. This complete set of rows returned by the statement is known as the result set. Some applications, especially interactive online applications, cannot always work effectively with the entire result set as a unit and require a mechanism to work with a small block of rows at a time. Cursors are an extension to result sets that provide that mechanism. Updateable cursors support data modification statements that update rows through the cursor. When positioned on a row in an updateable cursor, DELETE operations that target the base table rows used to build the current row in the cursor can be performed.

Q. 58
You are a database developer for an automobile dealership. You are designing a database to support a web site that will be used for purchasing automobiles. A person purchasing an automobile from the web site will be able to customize his or her order by selecting the model and color.

The manufacturer makes four different models of automobiles. The models can be ordered in any one of five colors. A default color is assigned to each model.

The models are stored in a table named Models, and the colors are stored in a table named Colors. These tables are shown in the exhibit.



You need to create a list of all possible model and color combinations. Which script should you use?

A.
```
SELECT m.ModelName, c.ColorName
FROM Colors AS c FULL OUTER JOIN Models AS m
    ON c.ColorID = m.ColorID
ORDER BY m.ModelName, c.ColorName
```

B.
```
SELECT m.ModelName, c.ColorName
FROM Colors AS c CROSS JOIN Models AS m
ORDER BY m.ModelName, c.ColorName
```

C.
```
SELECT m.ModelName, c.ColorName
FROM Colors AS m INNER JOIN Colors AS c
    ON m.ColorID = c.ColorID
ORDER BY m.ModelName, c.ColorName
```

D.
```
SELECT m.ModelName, c.ColorName
FROM Colors AS c LEFT OUTER JOIN Models AS m
    ON c.ColorID = m.ColorID
UNION
SELECT m.ModelName, c.ColorName
FROM Colors AS c RIGHT OUTER JOIN Models AS m
    ON c.ColorID = m.ColorID
ORDER BY m.ModelName, c.ColorName
```

E.
```
SELECT m.ModelName
FROM Models AS m
UNION
SELECT c.ColorName
FROM Colors AS c
ORDER BY m.ModelName
```

Answer: B.
Explanation: We want to produce all possible model and color combinations. Cross joins return all rows from the left table; each row from the left table is combined with all rows from the right table. Cross joins are also called Cartesian products.

Note: By using joins, we can retrieve data from two or more tables based on logical relationships between the tables. Joins can be specified in either the FROM or WHERE clauses and can be categorized as INNER JOIN, OUTER JOIN, or CROSS JOIN. Inner joins use a comparison operator to match rows from two tables based on the values in common columns from each table. Outer joins can be left, right, or full outer joins and are specified with one of the following sets of keywords when they are specified in the FROM clause: LEFT OUTER JOIN, RIGHT OUTER JOIN, or FULL OUTER JOIN. The result set of a left outer join includes all the rows from the left table specified in the LEFT OUTER clause, not just the ones in which the joined columns match. When a row in the left table has no matching rows in the right table, the associated result set row contains null values for all select list columns coming from the right table. A right outer join is the reverse of a left outer join. A full outer join returns all rows in both the left and right tables. Any time a row has no match in the other table, the select list columns from the other table contain null values. When there is a match between the tables, the entire result set row contains data values from the base tables.

Incorrect answers:

A: A FULL OUTER JOIN returns all rows in both the left and right tables. When a row has no match in the other table, the select list columns from the other table contain null values. When there is a match between the tables, the entire result set row contains data values from the base tables.

C: The inner join is also known as an equi-join. It returns all the columns in both tables, and returns only the rows for which there is an equal value in the join column.

D: The result set of a LEFT OUTER JOIN includes all the rows from the left table specified in the LEFT OUTER clause, not just the ones in which the joined columns match. When a row in the left table has no matching rows in the right table, the associated result set row contains null values for all select list columns coming from the right table. The result set of a RIGHT OUTER JOIN includes all the rows from the right table specified in the RIGHT OUTER clause, not just the ones in which the joined columns match. When a row in the right table has no matching rows in the left table, the associated result set row contains null values for all select list columns coming from the left table.

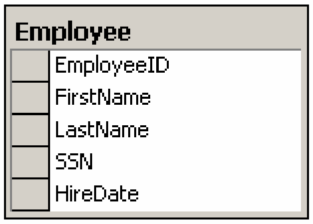
E: The UNION operator allows you to combine the results of two or more SELECT statements into a single result set. The result sets combined using UNION must all have the same structure. They must have the same number of columns, and the corresponding result set columns must have compatible data types.

Q. 59
You are a database developer for Adventure Works. A large amount of data has been exported from a human resources application to a text file. The format file that was used to export the human resources data is shown in the Format File exhibit.

Format File

```
1  SQLINT      0   4   "," 1  EmployeeID   ""
2  SQLCHAR     0   50  "," 2  Firstname    SQL_Latin1_General_CP1_AS
3  SQLCHAR     0   50  "," 3  Lastname SQL_Latin1_General_CP1_AS
4  SQLCHAR     0   10  "," 4  SSN      SQL_Latin1_General_CP1_AS
5  SQLDATETIME 0   8   ""  5  Hiredate ""
```

You need to import that data programmatically into a table named Employee. The Employee table is shown in the exhibit table.


Employee
EmployeeID
FirstName
LastName
SSN
HireDate

You need to run this import as quickly as possible. What should you do?

A. Use SQL-DMO and Microsoft Visual Basic Scripting Edition to create a **Table** object.
Use the **ImportData** method of the **Table** object to load the table.

B. Use SQL-DMO and Microsoft Visual Basic Scripting Edition to create a **Database** object.
Use the **CopyData** property of the **Database** object to load the table.

C. Use Data Transformation Services and Microsoft Visual Basic Scripting edition to create a **Package** object.
Create a **Connection** object for the text file.
Add a **BulkInsertTask** object to the **Package** object.
Use the **Execute** method of the **Package** object to load the data.

D. Use Data Transformation Services and Microsoft Visual Basic Scripting edition to create a **Package** object.
Create a **Connection** object for the text file.
Add an **ExecuteSQLTask2** object to the **Package** object.
Use the **Execute** method of the **ExecuteSQLTask2** object to load the data.

Answer: C.
Explanation: The BulkInsertTask object, based on the Transact-SQL BULK INSERT statement, provides the fastest method for copying large amounts of data from a text file to SQL Server. BulkInsertTask should be used for copying operations, and in situations where performance is the most important consideration. It is not used in conjunction with transformations during data import operations.

You can use a format file in the Bulk Insert task object.

The Execute method executes a Data Transformation Services (DTS) package or step.

Incorrect answers:

A: SQL-DMO is used for distributed queries, not for data transformation tasks.

Note: The ImportData method implements the bulk insert of data specified by the controlling BulkCopy object provided as an argument. However, SQL-DMO, which is short for SQL Server Distributed Management Objects, is the SQL Server Automation object model for SQL Server management. SQL-DMO objects, properties, methods, and collections can be used to write scripts and programs that administer multiple servers running SQL Server distributed across a network. Customized management needs can be met using SQL-DMO, or they can be integrated with SQL Server into other management tools. The power of the SQL-DMO object model for SQL Server management is enhanced through the use of a rapid application development language such as Visual Basic as a scripting environment for administrative tasks. SQL-DMO is thus not appropriate for this task.

B: SQL-DMO is used for distributed queries, not for data transformation tasks.

Note: The CopyData property controls data transfer from a source to a target database. However, SQL-DMO, which is short for SQL Server Distributed Management Objects, is the SQL Server Automation object model for SQL Server management. SQL-DMO objects, properties, methods, and collections can be used to write scripts and programs that administer multiple servers running SQL Server distributed across a network. Customized management needs can be met using SQL-DMO, or they can be integrated with SQL Server into other management tools. The power of the SQL-DMO object model for SQL Server management is enhanced through the use of a rapid application development language such as Visual Basic as a scripting environment for administrative tasks. SQL-DMO is thus not appropriate for this task.

D: The ExecuteSQLTask2 object allows you to execute a sequence of one or more SQL statements on a connection. However, we must specify which SQL Statements should be executed.

Q. 60
You are a database developer for an insurance company. The company has a database named Policies. You have designed stored procedures for this database that will use cursors to process large result sets. Analysts who use the stored procedures report that there is a long initial delay before data is displayed to them.

After the delay, performance is adequate. Only data analysts, who perform data analysis, use the Policies database.

You want to improve the performance of the stored procedures. Which script should you use?

A. EXEC sp_configure 'cursor threshold', 0

B. EXEC sp_dboption 'Policies' SET CURSOR_CLOSE_ON_COMMIT ON

C. SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

D. ALTER DATABASE Policies SET CURSOR_DEFAULT LOCAL

Answer: A.
Explanation: Use the cursor threshold option to specify the number of rows in the cursor set at which cursor keysets are generated asynchronously. If you set the cursor threshold to -1, all keysets are generated synchronously, which benefits small cursor sets. If you set the cursor threshold to 0, all cursor keysets are generated asynchronously. With other values, the query optimizer compares the number of expected rows in the cursor set and builds the keyset asynchronously if it exceeds the number set in the cursor threshold. Do not set the cursor threshold too low because small result sets are better built synchronously.

Note that cursor threshold is an advanced option. If you are using the sp_configure system stored procedure to change the setting, you can change the cursor threshold only when **show advanced options** is set to 1. The server-wide setting takes effect immediately (i.e. without a server stop and restart).

Incorrect answers:
B: The database is used for data analysis. The procedure is not likely to include any updates of the tables in the database, so the procedure would not contain any transactions. Changing the transaction configuration would not improve performance on the server.

Note: When CURSOR_CLOSE_ON_COMMIT is true, any cursors that are open when a transaction is committed or rolled back are closed. When false, such cursors remain open when a transaction is committed. When false, rolling back a transaction closes any cursors except those defined as INSENSITIVE or STATIC.

C: SERIALIZABLE is the highest level of transaction isolation, in which only one user can access a data row at a time. This solution is not relevant to the scenario. The database is used for data analysis and the transaction isolation level is not important since the data isn't changing.

D: Cursors are used to scroll through rowsets. This database is used for data analysis and cursors would not be used.

Note: When CURSOR_DEFAULT LOCAL is set, and a cursor is not defined as GLOBAL when it is created, the scope of the cursor is local to the batch, stored procedure, or trigger in which the cursor was created. The cursor name is valid only within this scope. The cursor can be referenced by local cursor variables in the batch, stored procedure, or trigger, or a stored procedure OUTPUT parameter. The cursor is implicitly deallocated when the batch, stored procedure, or trigger terminates, unless it was passed back in an OUTPUT parameter. If it is passed back in an OUTPUT parameter, the cursor is deallocated when the last variable referencing it is deallocated or goes out of scope.

Q. 61
You are a database developer for a bookstore. You are designing a stored procedure to process XML documents. You use the following script to create the stored procedure:

```
CREATE PROCEDURE spParseXML (@xmlDocument varchar(1000)) AS
DECLARE @docHandle int
EXEC sp_xml_preparedocument @docHandle OUTPUT, @xmlDocument
SELECT *
FROM OPENXML (@docHandle, '/ROOT/Category/Product',2)
WITH (ProductID int,
         CategoryID int,
         CategoryName varchar (50),
         [Description] varchar (50))
EXEC sp_xml_removedocument @docHandle
```

You execute this stored procedure and use an XML documents as the input document. The XML document is shown in the XML Document exhibit.

XML Document

```
<ROOT>
<Category CategoryID= "1" CategoryName= "General Books">
     <Product  ProductID="10248"  Description="Cooking  for
you">
     </Product>
</Category>
<Category CategoryID= "2" CategoryName= "Videos">
```

```
  <Product ProductID= "80248" Description= "7 Minute Abs">
     </Product>
</Category>
<Category CategoryID= "3" CategoryName= "Computer Books">
  <Product  ProductID=  "12345"  Description=  "Inside  SQL
Server 2000">
    </Product>
    <Product  ProductID=  "22345"  Description=  "Analysis
Services with SQL Server 2000">
     </Product>
</Category>
<ROOT>
```

You receive the output shown in the Output exhibit.

Output

```
ProductID  CategoryID  CategoryName       Description
NULL       NULL        NULL               NULL
NULL       NULL        NULL               NULL
NULL       NULL        NULL               NULL
NULL       NULL        NULL               NULL

(4 row(s) affected)
```

You need to replace the body of the stored procedure. Which script should you use?

A.
```
SELECT *
FROM OPENXML (@docHandle, '/ROOT/category/Product', 1)
WITH (ProductID int,
          CategoryID int,
          CategoryName varchar(50),
          [Description] varchar (50))
```

B.
```
SELECT *
FROM OPENXML (@docHandle, '/ROOT/category/Product', 8)
WITH (ProductID int,
             CategoryID int,
             CategoryName varchar(50),
             [Description] varchar (50))
```

C.
```
SELECT *
FROM OPENXML (@docHandle, '/ROOT/category/Product', 1)
WITH (ProductID int,
            CategoryID int,
            CategoryName varchar(50), '@CategoryName',
            [Description] varchar (50))
```

D.
```
SELECT *
FROM OPENXML (@docHandle, '/ROOT/category/Product', 1)
WITH (ProductID int,
            CategoryID int '../@CategoryID',
            CategoryName varchar(50), '../@CategoryName',
            [Description] varchar (50))
```

Answer: D.
Explanation:
The syntax of the OPEN XML statement is

```
OPENXML(idoc int [in],rowpattern nvarchar[in],[flags byte[in]])
[WITH (SchemaDeclaration | TableName)]
```

We want the flags option to be set to 1. We want the XML attributes to map to the columns in the rowset. Then we must specify the correct schema declaration using the WITH option.

By studying the XML document exhibit we see that the CategoryID and CategoryName in the rowset map to the attributes of the parent of the nodes identified by rowpattern in the XML document. In short, the CategoryID and CategoryID attributes are defined at a higher level than ProductID and Description attributes in the XML document. We supply this information in the schema declaration by using:

```
WITH (ProductID int,
CategoryID int '../CategoryID',
CategoryName varchar(50), '../CategoryName',
                        [Description] varchar (50))
```

Note '../CategoryID' and '../CategoryName', which map to the parent of the nodes.

Incorrect answers:

A: The schema declaration is incorrectly flat, but from the XML documents exhibit we see that CategoryID and CategoryName map to attributes of parent node.

B: The flag should be set to 1, not to 8. The schema declaration is also incorrect.

C: The schema declaration is incorrect. CategoryID and CategoryName map to attributes of parent node, and the schema declaration must provide this information.

Q. 62
You are a database developer for Adventure Works. You are designing a script for the human resources department that will report yearly wage information. There are three types of employee. Some employees earn an hourly wage, some are salaried, and some are paid commission on each sale that they make. This data is recorded in a table named Wages, which was created by using the following script:

```
CREATE TABLE Wages
        (
        emp_id tinyint identity,
        hourly_wage decimal NULL,
        salary decimal NULL,
        commission decimal NULL,
        num_sales tinyint NULL
        )
```

An employee can have only on type of wage information. You must correctly report each employee's yearly wage information.

Which script should you use?

A.
```
SELECT CAST (hourly_wage * 40 * 52 +
salary +
commission * num_sales AS MONEY) as YearlyWages
FROM Wages
```

B.
```
SELECT CAST (COALESCE (hourly_wage * 40 * 52,
Salary,
commission * num_sales) AS MONEY) as YearlyWages
FROM Wages
```

C.
```
SELECT CAST (CASE
WHEN((hourly_wage,) IS NOTNULL) THEN hourly_wage * 40 * 52
WHEN(NULLIF(salary,NULL)IS NULL)THEN salary
ELSE commission * num_sales
END
AS MONEY)
As YearlyWages
FROM Wages
```

D.
```
SELECT CAST(CASE
            WHEN (hourly_wage IS NULL)THEN salary
            WHEN (salary IS NULL)THEN commission*num_sales
            ELSE commission * num_sales
            END
        AS MONEY)
        As YearlyWages
        FROM Wages
```

Answer: B.
Explanation: We need make a calculation from four columns of the table. The problem is that all four columns allow NULL values. The COALESCE function comes in handy as it returns the first non-null expression in its argument line. The following COALESCE statement produces the first non-null wage item:

```
COALESCE (hourly_wage * 40 * 52,
Salary,
commission * num_sales)
```

The CAST clause converts an expression of one data type to another.

Reference: BOL, COALESCE

Incorrect answers:

A: This code adds the calculations of the hourly_wage, salary and the commission sales. The problem is that NULL + anything equals NULL. The calculation would always produce a NULL value.

C: The CASE function evaluates a list of conditions and returns one of multiple possible result expressions. The second line in this code of this solution converts the value in the hourly_wage columns if the column is not null. However, the third line does not do the same for the salary column. In this line the NULLIF statement is used. This statement returns a null value if the two specified expressions are equivalent. It thus does not return the yearly wage value for salaried workers.

D: The CASE function evaluates a list of conditions and returns one of multiple possible result expressions. However, the second line,

```
WHEN (hourly_wage IS NULL)THEN salary
```

is not logically correct. Both the hourly_wage and the salary might be NULL, and a NULL value could be produced.

Q. 63
You are a database developer for an insurance company. The company's regional offices transmit their sales information to the company's main office in XML documents. The XML documents are then stored

in a table named SalesXML, which is located in a SQL Server 2000 database. The data contained in the XML documents includes the names of insurance agents, the names of insurance policy owners, information about insurance policy beneficiaries, and other detailed information about the insurance policies. You have created tables to store information extracted from the XML documents.

You need to extract this information from the XML documents and store it in the tables. What should you do?

A. Use SELECT statements that include the FOR XML AUTO clause to copy the data from the XML documents into the appropriate tables.

B. Use SELECT statements that include the FOR XML EXPLICIT clause to copy the data from the XML documents into the appropriate tables.

C. Use the OPENXML function to access the data and to insert it into the appropriate tables.

D. Build a view on the **SalesXML** table that displays the contents of the XML documents. Use SELECT INTO statements to extract the data from this view into the appropriate tables.

Answer: C.
Explanation: The Transact-SQL OPENXML function can be used to insert data represented as an XML document. OPENXML is a rowset provider similar to a table or a view, providing a rowset over in-memory XML documents. OPENXML allows access to XML data as if it is a relational rowset by providing a rowset view of the internal representation of an XML document. The records in the rowset can be stored in database tables. OPENXML can be used in SELECT, and SELECT INTO statements where a source table or view can be specified

Incorrect answers:
A: FOR XML in a SELECT clause produces XML data from rowsets, but we want to produce rowsets from XML data.

Note: The FOR XML of the SELECT statement is used to execute SQL queries directly, i.e. not from within stored procedures, to return results as XML rather than standard rowsets. In this scenario, we have to extract data from a XML document. The XML mode can be specified in the FOR XML clause. The AUTO mode returns query results as nested XML elements, listing each table in the FROM clause from which at least one column is listed in the SELECT clause. The columns listed in the SELECT clause are mapped to the appropriate attributes of the elements. By default, AUTO mode maps the table columns to XML attributes.

B: FOR XML in a SELECT clause produces XML data from rowsets, but we want to produce rowsets from XML data.

Note: The FOR XML of the SELECT statement is used to execute SQL queries directly, i.e. not from within stored procedures, to return results as XML rather than standard rowsets. In this scenario data we have to extract data from a XML document. In the EXPLICIT mode, the shape of the XML document returned by the execution of the query conforms to the design indicated by the query writer. Here the query must be written in such a way that the additional information about the expected nesting is explicitly specified as part of the query.

D: SQL Server 2000 cannot build views based on XML documents without first querying the XML document using the OPENXML. Furthermore, SELECT INTO creates new tables. We want to insert the data into existing tables.

Q. 64
You are a database developer for an insurance company. You create a table named Insured, which will contain information about persons covered by insurance policies. You use the script shown in the exhibit to create this table.

```
CREATE TABLE dbo.Insured (
         InsuredID int IDENTITY (1, 1)  NOT NULL,
         PolicyID int NOT NULL,
         InsuredName char(30) NOT NULL,
         InsuredBirthDate datetime NOT NULL,
        CONSTRAINT PK_Insured PRIMARY KEY CLUSTERED
        (
                  InsuredID
        ),
        CONSTRAINT FK_Insured_Policy FOREIGN KEY
        (
                   PolicyID
        ) REFERENCES dbo.Policy(
                   PolicyID
        )
)
```

A person covered by an insurance policy is uniquely identified by his or her name and birth date. An insurance policy can cover more than one person. A person cannot be covered more than once by the same insurance policy.

You must ensure that the database correctly enforces the relationship between insurance policies and the persons covered by insurance policies. What should you do?

A. Add the **PolicyID**, **InsuredName**, and **InsuredBirthDate** columns to the primary key

B. Add a UNIQUE constraint to enforce the uniqueness of the combination of the **PolicyID**, **InsuredName**, and **InsuredBirthDate** columns.

C. Add a CHECK constraint to enforce the uniqueness of the combination of the **PolicyID**, **InsuredName**, and **InsuredBirthDate** columns.

D. Create a clustered index on the **PolicyID**, **InsuredName**, and **InsuredBirthDate** columns.

Answer: B.
Explanation: UNIQUE constraints enforce the uniqueness of the values in a set of columns and ensure that no duplicate values are entered in specific columns that do not participate in a PRIMARY KEY. Primary keys also enforce uniqueness, but they do not allow null values. Although both UNIQUE constraints and PRIMARY KEY constraints enforce uniqueness, UNIQUE constraints are recommended over PRIMARY KEY constraints when you want to enforce the uniqueness of a column, or combination of columns, that is not the primary key, as multiple UNIQUE constraints can be defined on a table, while only one PRIMARY KEY constraint can be defined on a table.

Incorrect answers:
A: We only want to enforce uniqueness. We don't have to use a PRIMARY KEY constraint. Unless some specific performance gain is achieved by adding the column to the primary key, we should use an UNIQUE constraint instead. Too many columns in an index would decrease performance.

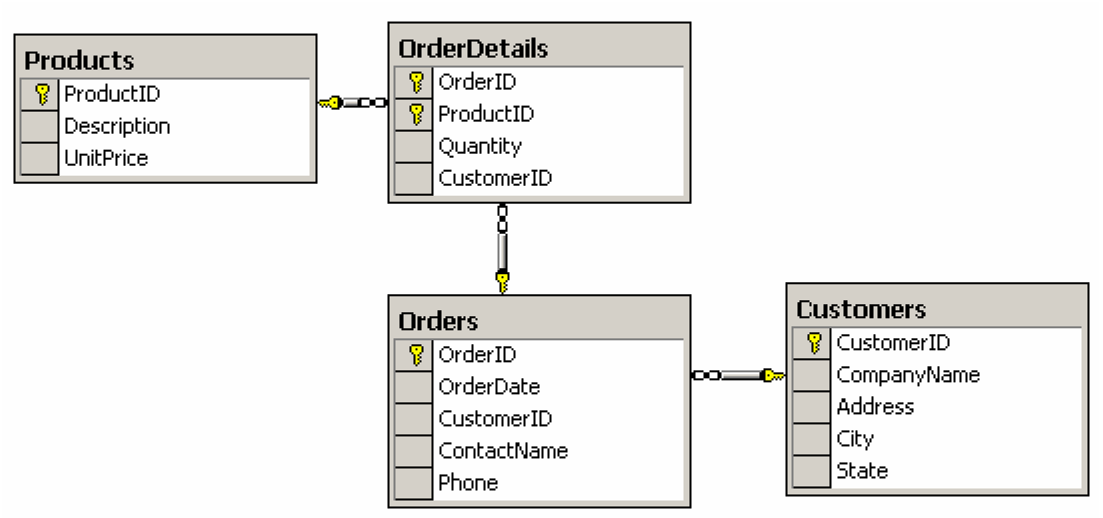
Note: PRIMARY KEY constraints identify the column or set of columns whose values uniquely identify a row in a table. No two rows in a table can have the same PRIMARY KEY value or NULL values. Using a small, integer column as a primary key is recommended and each table should have a PRIMARY KEY. A table may have more than one combination of columns that could uniquely identify the rows in a table; each combination is a candidate key of which one is chosen to be the primary key.

C: CHECK constraints cannot be used to force uniqueness in a column. They enforce domain integrity by limiting the values that can be placed in a column. A CHECK constraint specifies a Boolean search condition that is applied to all values entered for the column. All values that do not evaluate to TRUE are rejected.

D: There can only be one clustered index in a table, and in this table the primary key is clustered. We cannot add another clustered index.

Note: A clustered index determines the physical order of data in a table. As a result, a table can contain only one clustered index, but the index can comprise multiple columns. A clustered index is particularly efficient on columns that are often searched for ranges of values. After the row with the first value is found using the clustered index, rows with subsequent indexed values are guaranteed to be physically adjacent.

Q. 65
You are designing a database for Tailspin Toys. 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?

A. Create a new table named **CustomerContact.**
Add **CustomerID**, **ContactName**, and **Phone** columns to this table.

B. Create a new composite PRIMARY KEY constraint on the **OrderDetails** table.
Include the **OrderID**, **ProductID**, and **CustomerID** columns in the constraint.

C. Remove the PRIMARY KEY constraint from the **OrderDetails** table.
Use an IDENTITY column to create a surrogate key for the **OrderDetails** table.

D. Remove the **CustomerID** column from the **OrderDetails** table.

E. Remove the **Quantity** column from the **OrderDetails** table.
Add a **Quantity** column to the **Orders** table.

Answer: D.
Explanation: A normalized database design is the best starting point for an efficient design. By examining the exhibit we see that the CustomerID column in the OrderDetails table looks out of place. The CustomerID column belongs to the Customer table and to the tables that reference the CustomerID column in the Customers table (here, the Orders table) with a foreign key constraint. Removing the CustomerID column from the OrderDetails table would normalize the design, reduce redundancy, and improve performance.

Incorrect answers:
A: Adding a new table would increase the possibilities to store data, but it would not improve query performance. We should normalize the design before adding more tables.

B: The primary key of the OrderDetails table already uniquely defines the rows in the table. Adding another column to the primary key would increase the size of the index. This would most likely decrease performance.

Note: When a PRIMARY KEY constraint is specified for a table, SQL Server 2000 enforces data uniqueness by creating a unique index for the primary key columns. This index permits fast access to data when the primary key is used in queries. If a PRIMARY KEY constraint is defined on more than one column, values may be duplicated within one column, but each combination of values from all the columns in the PRIMARY KEY constraint definition must be unique.

C: Removing the PRIMARY KEY constraint from the OrderDetails table and using an IDENTITY column to create a surrogate key for the OrderDetails table would not improve query times on this table. It is thus inappropriate

E: Removing the Quantity column from the OrderDetails table and adding it to the Orders table would denormalize the database design. Quantity is an attribute of the OrderDetails table (quantity of ordered items), not an attribute of Orders.

Q. 66
You are a database developer for an automobile dealership. The company stored its automobiles inventory data in a SQL Server 2000 database. Many of the critical queries in the database join three tables named Make, Model, and Manufacturer. These tables are updated infrequently.

You want to improve the response time of the critical queries. What should you do?

A. Create an indexed view on the tables.
B. Create a stored procedure that returns data from the tables.
C. Create a scalar user-defined function that returns data from the tables.
D. Create a table-valued user-defined function that returns data from the tables.

Answer: A.
Explanation: Indexes are used to gain fast access to specific information in a table. An index is a structure that orders the values of one or more columns in a database table. The index provides pointers to the data values stored in specified columns of the table, and then orders those pointers according to the sort order specific to the type of index. Views are sometimes called virtual tables because the result set returned by the view has the same general form as a table with columns and rows, and can be indexed and referenced the same way as tables.

Note: Index views
For a standard view, the overhead of dynamically building the result set for each query that references the view can be substantial if the view involves complex processing of large numbers of rows (such as aggregating large amounts of data) or joining many rows. If such views are frequently referenced in queries, you can improve performance by creating a unique clustered index on the view. When a unique clustered index is created on a view, the view is executed and the result set is stored in the database in the same way that a table with a clustered index is stored.

Existing queries can benefit from the improved efficiency of retrieving data from the indexed view without having to be recoded.

Incorrect answers:

B: There would no significant performance improvement gained by creating a stored procedure out of the SQL query. In SQL Server 2000 batches also have execution plans that are stored.

Note: A stored procedure is a group of Transact-SQL statements that are compiled into a single execution plan. SQL Server 2000 stored procedures can return data as output parameters, which can return either data or a cursor variable; as return 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. 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.

C: Functions in programming languages are subroutines used to 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. Scalar functions return a single data value of the type defined in a RETURNS clause. All scalar data types, including bigint and sql_variant, can be used. The return type can be any data type except text, ntext, image, cursor, and timestamp. Scalar functions do not return tables.

D: Moving the SQL queries inside a function would not improve performance.

Note: Functions in programming languages are subroutines used to 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. Table-valued functions return a table. Such functions can be used where table or view expressions are allowed in Transact-SQL queries. While views are limited to a single SELECT statement, user-defined functions can contain additional statements that allow more powerful logic than is possible in views.

Q. 67
You are a database developer for an insurance company. The company has one main office and 18 regional offices. Each office has one SQL Server 2000 database. The regional offices are connected to the main office by a high-speed network.

The main office database is used to consolidate information from the regional office databases. The tables in the main office database are partitioned horizontally. The regional office location is used as part of the primary key for the main office database. You are designing the physical replication model.

What should you do?

A. Configure the main office as a publishing Subscriber.

B. Configure the main office as a publisher with a remote distributor.

C. Configure the main office as a central publisher and the regional offices as Subscribers.

D. Configure the regional offices as Publishers and the main office as a central Subscriber.

Answer: D.
Explanation: In this scenario the main office consolidates data from the regional offices. Hence, data needs to be replicated to the main office. In this event the regional offices are the publishers and the main office the subscriber. This is an example of merge replication.

Note: The simplest SQL Server 2000 replication model has a single Publisher and Distributor on the same server and one Subscriber on another server. This becomes more complex as more Subscribers are added to the Publisher and Distributor. The Publisher is a server that makes data available for replication to other servers. In addition to being the server where you specify which data is to be replicated, the Publisher also detects which data has changed and maintains information about all publications at that site. Subscribers are servers that receive replicated data. Subscribers subscribe to publications, not to individual articles within a publication, and they subscribe only to the publications that they need.

Incorrect answers:

A: A server cannot subscribe to its own publication**.**

B: The Publisher is a server that makes data available for replication to other servers, while the Distributor is a server that contains the distribution database and stores meta data, history data, and/or transactions. The Distributor can be a separate server from the Publisher or it can be the same server as the Publisher. A remote Distributor is a computer that is physically separate from the Publisher and is configured as a Distributor of replication. A local Distributor is a computer that is configured to be both a Publisher and a Distributor of replication.

C: In this scenario the main office consolidates data from the regional offices. Hence data needs to be replicated from the regional offices to the main office. In this event the regional offices are the publishers and the main office the subscriber.

Q. 68

You are a database developer for a clothing retailer. The company has a database named Sales. This database contains a table named Inventory. The Inventory table contains the list of items for sale and the quantity available for each of those items. When sales information is inserted into the database, this table is updated. The stored procedure that updates the Inventory table is shown in the exhibit.

```
CREATE PROCEDURE UpdateInventory @IntID int
AS
BEGIN

DECLARE @Count int

BEGIN TRAN

SELECT @Count = Available
FROM Inventory WITH (HOLDLOCK)
WHERE InventoryID = @IntID

IF (@Count > 0)
UPDATE Inventory SET Available = @Count - 1
WHERE InventoryID = @IntID

COMMIT TRAN

END
```

When this procedure executes, the database server occasionally returns the following error message:

```
Transaction (Process ID 53) was deadlocked on {lock} resources with another
process and has been chosen as the deadlock victim. Rerun the transaction.
```

You need to prevent the error message from occurring while maintaining data integrity. What should you do?

A. Remove the table hint.

B. Change the table hint to UPDLOCK.

C. Change the table hint to REPEATABLEREAD.

D. Set the transaction isolation level to SERIALIZABLE.

E. Set the transaction isolation level to REPEATABLE READ.

Answer: B.
Explanation: This is a deadlock problem. We must resolve this problem. The SQL batch of this scenario basically consists of an SELECT statement and an UPDATE statement. The SELECT statement includes a table hint: WITH (HOLDLOCK).

This table hint is very restrictive. The rows are completely locked. We need to remove this restrictive table hint and replace it with the table hint UPDLOCK, which is less restrictive than HOLDLOCK. It allows reads of the rows, but no updates.

Note: table hint
A table hint specifies that a table scan, or one or more indexes, must be used by the query optimizer, or a locking method must be used by the query optimizer with this table and for this SELECT. The query optimizer can usually pick the best optimization method without hints being specified, therefore it is recommended that hints only be used as a last resort by experienced developers and database administrators.

Incorrect answers:
A: If we just remove the table hint, we will have the default isolation of REPEATABLE UNCOMMITED, but that would allow, for example, phantom records. And at the time the UPDATE statement executed, the rows could have changed. We cannot allow that.

C: The REPEATABLEREAD table hint specifies that a scan be performed with the same locking semantics as a transaction running at REPEATABLE READ isolation level. The REPEATABLE READ isolation level could result in phantom records. This cannot be tolerated in this scenario.

D: An isolation level determines the degree to which data is isolated for use by one process and guarded against interference from other processes. The SERIALIZABLE isolation level is the highest level of isolation and completely blocks other users or applications from accessing data that has been accessed by the transaction until the transaction is completed. It would most likely cause more locks and deadlocks. The number of error messages would increase, not decrease.

E: An isolation level determines the degree to which data is isolated for use by one process and guarded against interference from other processes. The REPEATABLE READ isolation level could result in phantom records. This cannot be tolerated in this scenario.

Q. 69
You are a database developer for Wide World Importers. The company tracks its order information in a SQL Server 2000 database. The database includes two tables that contain order details. The tables are named Order and LineItem. The script that was used to create these tables is shown in the exhibit.

```
CREATE TABLE dbo.Order
     (
     OrderID int NOT NULL,
     CustomerID int NOT NULL,
     OrderDate datetime NOT NULL,
     CONSTRAINT DF_Order_OrderDate DEFAULT (getdate())FOR OrderDate,
     CONSTRAINT PK_Order PRIMARY KEY CLUSTERED (OrderID)
     )

CREATE TABLE dbo.LineITEM
     (
     ItemID int NOT NULL,
     OrderID   INT NOT NULL,
     ProductID int  NOT NULL,
     Price money NOT NULL,
     CONSTRAINT PK_LineITEM PRIMARY KEY CLUSTERED (ItemID),
     CONSTRAINT FK_LineITEM_Order FOREIGN KEY (OrderID)
          REFERENCES dbo.Order (OrderID)
     )
```

The company's auditors have discovered that every item that was ordered on June 1, 2000, was entered with a price that was $10 more than its actual price. You need to correct the data in the database as quickly as possible.

Which script should you use?

A.
```
UPDATE l
SET Price = Price – 10
FROM LineItem AS l INNER JOIN [Order] AS o
    ON l.OrderID = o.OrderID
WHERE o.OrderDate >= ‘6/1/2000’
AND o.OrderDate < ‘6/2/2000’
```

B.
```
UPDATE l
SET Price = Price – 10
FROM LineItem AS l INNER JOIN [Order] AS o
    ON l.OrderID = o.OrderID
WHERE o.OrderDate = ‘6/1/2000’
```

C.
```
DECLARE @ItemID int

DECLARE items_cursor CURSOR FOR
SELECT l.ItemID
FROM LineItem AS l INNER JOIN [Order] AS o
    ON l.OrderID = o.OrderID
WHERE o.OrderDate >= ‘6/1/2000’
AND o.OrderDate < ‘6/1/2000’
FOR UPDATE

OPEN items_cursor
FETCH NEXT FROM items_cursor INTO @ItemID

WHILE @@FETCH_STATUS = 0
    BEGIN
            UPDATE LineItem SET Price = Price – 10
            WHERE CURRENT OF items_cursor

            FETCH NEXT FROM items_cursor INTO @ItemID
    END

CLOSE items_cursor
DEALLOCATE items_cursor
```

D.
```
DECLARE @OrderID int

DECLARE order_cursor CURSOR FOR
SELECT ordered FROM [Order]
WHERE OrderDate = ‘6/1/2000’
```

```
OPEN order_cursor
FETCH NEXT FROM order_cursor INTO @OrderID

WHILE @@FETCH_STATUS = 0
BEGIN
    UPDATE LineItem SET Price = Price – 10
    WHERE OrderID= @OrderID

    FETCH NEXT FROM order_cursor INTO @OrderID
END

CLOSE order_cursor
DEALLOCATE order_cursor
```

Answer: A.
Explanation: We use a simple SQL query that makes the required update of the table. If possible, we prefer a simple T-SQL statement compared to a cursor. Furthermore, we need to include a test for the correct date of the order. The OrderDate column uses the datetime data type. The datetime datatype represents both the day and the time of the date; therefore, we cannot use a simple equality test. We have to use the following WHERE clause to capture the exact date:

```
WHERE o.OrderDate >= ‘6/1/2000’
AND o.OrderDate < ‘6/2/2000’
```

Note: We examine the SQL Query in detail:

```
UPDATE l
```

This line updates the LineItem table. The alias is defined in the FROM clause below.

```
SET Price = Price - 10
```

The price column is decreased by 10. Note that NULL values are not allowed in the price column. We don't need a cursor to check for NULL values in the Price column.

```
FROM LineItem AS l INNER JOIN [Order] AS o
  ON l.OrderID = o.OrderID
```

The LineItem table is joined with the Order table on the OrderID column.

```
WHERE o.OrderDate >= ‘6/1/2000’
AND o.OrderDate < ‘6/2/2000’
```

We restrict the rows to rows with an OrderDate of 6/1/2000. Note that OrderDate both contains the date and the time of the day. We must therefore test for a time interval, not a single point of time.

Reference: BOL, datetime and smalldatetime

Incorrect answers:

B: The OrderDate column has the datetime datatype. It contains both the date and the time of the day. We must therefore test for a time interval, not a single point in time.

C: This proposed solution works, but it is not the fastest way to achieve the goal. Cursors are not as efficient as SQL queries and require more system resources. Cursors should only be used if there is no better solution available.

D: This solution has an incorrect WHERE clause. (It only tests for a single point in time, but we need to test for an interval.) It would also be better to use a T-SQL statement instead of a cursor.

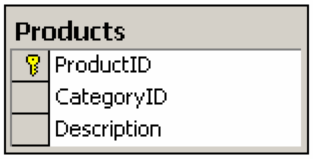
Q. 70

You are a database developer for a bookstore. Each month, you receive new supply information from your vendors in the form of an XML document. The XML document is shown in the XML Document exhibit.

XML Document

```
<ROOT>
<CategoryID= "2" CategoryName= "Videos">
      <Product ProductID= "80248" Description= "7 Minute Abs">
      </Product>
</Category>
<Category CategoryID= "3" CategoryName= "Computer Books">
  <Product ProductID= "12345" Description= "Inside SQL Server 2000">
     </Product>
</Category>
<Category CategoryID= "3" CategoryName= "Computer Books">
  <Product  ProductID=  "22345"  Description=  "Analysis  Services  with  SQL
Server 2000">
    </Product>
</Category>
<ROOT>
```

You are designing a stored procedure to read the XML document and to insert the data into a table named Products. The Products table is shown in the Products Table exhibit.



Which script should you use to create this stored procedure?

A.
```
CREATE PROCEDURE spAddCatalogItems (
                              @xmlDocument varchar (8000))

AS

BEGIN
DECLARE @docHandle int
EXEC sp_xml_preparedocument @docHandle OUTPUT, @xmlDocument
INSERT INTO Products
EXEC sp_xml_preparedocument @docHandle OUTPUT, @xmlDocument
INSERT INTO Products
SELECT * FROM
OPENXML (@docHandle, '/ROOT/Category/Product', 1)
WITH Products

EXEC sp_xml_removedocument @docHandle
END
```

B.
```
CREATE PROCEDURE spAddCatalogItems (
                              @xmlDocument varchar (8000))

AS

BEGIN
DECLARE @docHandle int
EXEC sp_xml_preparedocument @docHandle OUTPUT, @xmlDocument

INSERT INTO Products
SELECT * FROM OPENXML (@docHandle, '/ROOT/Category/Product', 1)
WITH (ProductID int './@ProductID',
        CategoryID int '../@CategoryID',
        [Description] varchar (100)  './@Description')

EXEC sp_xml_removedocument @docHandle
END
```

C.

```
CREATE PROCEDURE spAddCatalogItems (
                        @xmlDocument varchar (8000))
AS

BEGIN
INSERT INTO Products
SELECT * FROM OPENXML (
@xmlDocument, '/ROOT/Category/Product', 1)
WITH (ProductID int, Description varchar (50))
END
```

D.

```
CREATE PROCEDURE spAddCatalogItems (
        @xmlDocument varchar (8000))
AS

BEGIN
INSERT INTO Products
SELECT* FROM
OPENXML (@xmlDocument, '/ROOT/Category/Product',1)
WITH Products
END
```

Answer: B.
Explanation: The OPENXML function is a Transact-SQL keyword that provides a rowset over in-memory XML documents. OPENXML enables access to XML data as if it were a relational rowset by providing a rowset view of the internal representation of an XML document. OPENXML can be used in SELECT and SELECT INTO statements wherever a rowset provider, such as a table, a view, or OPENROWSET, can appear as the source.

To write queries against an XML document by using OPENXML, you must first call the sp_xml_preparedocument system stored procedure, which parses the XML document and returns a handle to the parsed document that is ready for consumption.

The syntax of the OPENXML is:

```
OPENXML(idoc int [in],rowpattern nvarchar[in],[flags byte[in]])
[WITH (SchemaDeclaration | TableName)]
```

The WITH (SchemaDeclaration | TableName) specifies a Schema Declaration.

Note: XML is a hypertext programming language used to describe the contents of a set of data and how the data should be output to a device or displayed on a Web page.

Incorrect answers:

A: There is an error in the OPENXML statement:

OPENXML (@docHandle, '/ROOT/Category/Product', 1)
WITH Products

The **WITH Products** option should supply a schema declaration but Products is just an ordinary SQL server table. The OPENXML statement wouldn't run.

C: The sp_xml_preparedocument system stored procedure must first parse the XML document. You can't directly open a XML document with the OPENXML command.

D: The sp_xml_preparedocument system stored procedure must first parse the XML document. You can't directly open a XML document with the OPENXML command.

Q. 71

You are a database developer for Trey Research. You are designing a SQL Server 2000 database that will be distributed with an application to numerous companies. You create several stored procedures in the database that contain confidential information. You want to prevent the companies from viewing this confidential information.

What should you do?

A. Remove the text of the stored procedures from the **syscomments** system table.

B. Encrypt the text of the stored procedures.

C. Deny SELECT permissions on the **syscomments** system table to the **public** role.

D. Deny SELECT permissions on the **sysobjects** system table to the **public** role.

Answer: B.

Explanation: The WITH ENCRYPTION clause can be used to ensure that the stored procedure's definition cannot be viewed by other users. The procedure definition is then stored in an unreadable form. When a stored procedure is encrypted, its definition cannot be decrypted and cannot be viewed by anyone, including the owner of the stored procedure or the system administrator.

Incorrect answers:

A: The syscomments system table contains entries for each view, rule, default, trigger, CHECK constraint, DEFAULT constraint, and stored procedure. The text column contains the SQL definition statements, which are limited to a size of 4 MB. This table is stored in each database. None of the entries in syscomments should be deleted, as the corresponding stored procedure will not function properly when its entry in syscomments is manually removed or modified. To hide or encrypt stored procedure definitions, use CREATE PROCEDURE (or ALTER PROCEDURE) with the ENCRYPTION keyword.

C: The syscomments system table contains entries for each view, rule, default, trigger, CHECK constraint, DEFAULT constraint, and stored procedure. Denying SELECT permission for the public role would not hide the procedure definition from users.

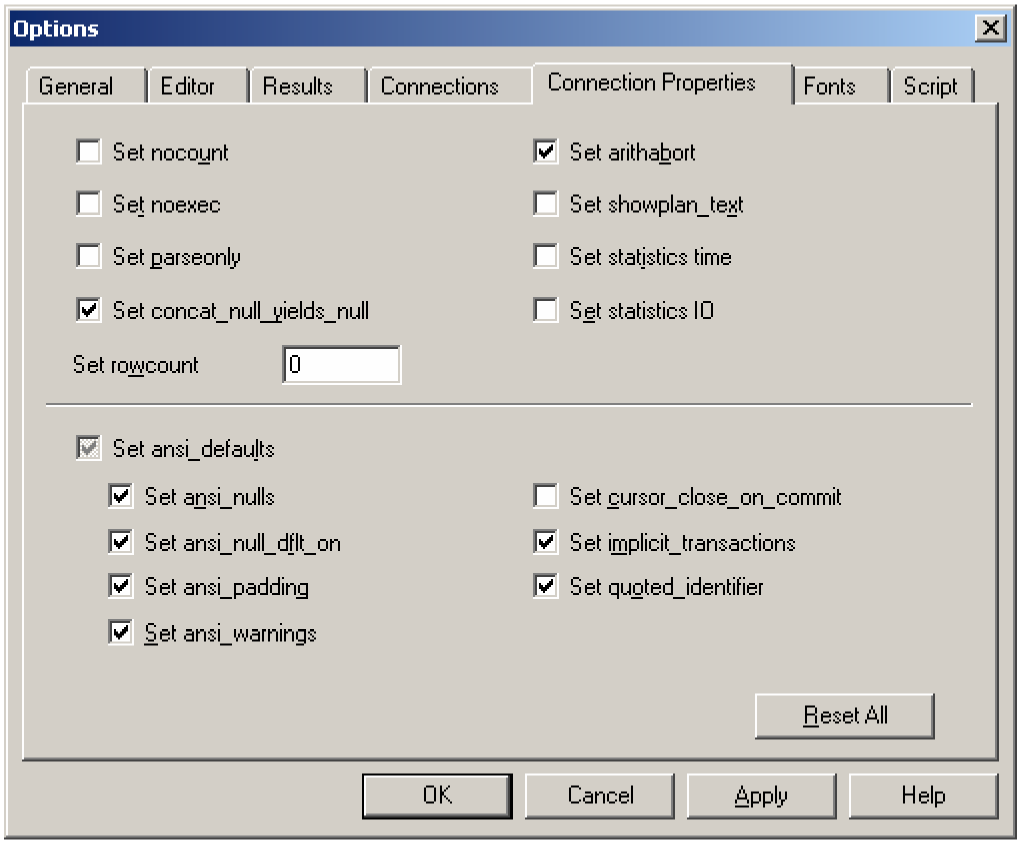
D: The sysobjects system table contains one row for each object (constraint, default, log, rule, stored procedure, and so on) created within a database. Denying SELECT permission for the public role would not hide the procedure definition from users.

Q. 72
You are a database developer for Southridge Video. The company stores its sales information in an SQL Server 2000 database. You are asked to delete order records from this database that are more than five years old. To delete the records, you execute the following statement in SQL Query Analyzer:

```
DELETE FROM Orders WHERE OrderDate < (DATEADD (year, -5, getdate()))
```

You close SQL Query Analyzer after you receive the following message:
```
(29979 row(s) affected)
```

You examine the table and find that the old records are still in the table. The current connection properties are shown in the exhibit.



- [ ] Set nocount
- [ ] Set noexec
- [ ] Set parseonly
- [✓] Set concat_null_yields_null
- [✓] Set arithabort
- [ ] Set showplan_text
- [ ] Set statistics time
- [ ] Set statistics IO

Set rowcount 0

- [✓] Set ansi_defaults
- [✓] Set ansi_nulls
- [✓] Set ansi_null_dflt_on
- [✓] Set ansi_padding
- [✓] Set ansi_warnings
- [ ] Set cursor_close_on_commit
- [✓] Set implicit_transactions
- [✓] Set quoted_identifier

What should you do?

A. Delete records in tables that reference the Orders table.

B. Disable triggers on the Orders table.

C. Execute a SET IMPLICIT_TRANSACTIONS OFF statement.

D. Execute a SET CURSOR_CLOSE_ON_COMMIT ON statement.

E. Change the logic in the DELETE statement.

Answer: C.
Explanation: Looking at the exhibit, we see that the **SET IMPLICIT_TRANSACTIONS** setting is enabled. This is not the default setting. When the SET IMPLICIT_TRANSACTIONS is ON, the connection set in implicit transaction mode. When OFF, the connection is set in autocommit transaction mode.

When a connection is in implicit transaction mode and the connection is not currently in a transaction, executing any ALTER TABLE, CREATE, DELETE, DROP, FETCH, GRANT, INSERT, OPEN, REVOKE, SELECT, TRUNCATE TABLE, or UPDATE statements starts a transaction. Transactions that are automatically opened by these statements as the result of this setting being ON must be explicitly committed by the user at the end of the transaction. Otherwise, the transaction and all the data changes it contains are rolled back when the user disconnects. This describes what has happened in this scenario.

Note: The SET IMPLICIT_TRANSACTIONS default setting is OFF.

Incorrect answers:

A: Deleting rows without committing the implicit transaction will not delete any rows at all until the transaction is committed; only a message, "*xxxx* **rows affected**", will be displayed, exactly as in the scenario.

B: An INSTEAD OF DELETE trigger could prevent any deletions on a table. But before we try disabling triggers we should attack the main problem; we should disable the IMPLICIT_TRANSACTIONS setting.

Note: When an INSTEAD OF trigger is defined on DELETE actions against a table or view, the normal action of the DELETE statement is replaced by the execution of trigger.

D: The main problem in this scenario is the IMPLICT_TRANSACTIONS setting. This setting should be disabled before we try to find other possible problems, such as trigger settings.

Note: When SET CURSOR_CLOSE_ON_COMMIT is ON, any open cursors are closed automatically when a transaction is committed. By default, this setting is OFF and cursors remain open across transaction boundaries, closing only when the connection is closed or when they are explicitly closed. However, a cursor is not being used in this scenario. This solution is thus not relevant.

E: The DATEADD function adds an interval to a date you specify. YEAR returns an integer that represents the year part of a specified date. GETDATE returns the current system date and time in the SQL Server standard internal format for datetime values. This value is compared to the OrderDate column in the Orders table. The delete statement is correct and not flawed in any way.

Q. 73
You are a database developer for a telemarketing company. You are designing a database named CustomerContacts. This database will be updated frequently. The database will be about 1 GB in size. You want to achieve the best possible performance for the database. You have 5 GB of free space on drive C.

Which script should you use to create the database?

A.
```
CREATE DATABASE CustomerContacts
ON
(NAME = Contacts_dat,
FILENAME = ‘c:\data\contacts.mdf’,
SIZE = 10,
MAXSIZE = 1GB
FILEGROWTH= 5)
```

B.
```
CREATE DATABASE CustomerContacts
ON
(NAME = Contacts_dat,
FILENAME = ‘c:\data\contacts.mdf’,
SIZE = 10,
MAXSIZE = 1GB
FILEGROWTH= 10%)
```

C.
```
CREATE DATABASE CustomerContacts
ON
(NAME = Contacts_dat,
FILENAME = ‘c:\data\contacts.mdf’,
SIZE = 100,
MAXSIZE = UNLIMITED)
```

D.
```
CREATE DATABASE CustomerContacts
ON
(NAME = Contacts_dat,
FILENAME = ‘c:\data\contacts.mdf’,
SIZE = 1GB)
```

Answer: D.
Explanation: We should create a database that has a size close to the estimated database size of 1GB. The database will be created by the T-SQL CREATE DATABASE command.

The SIZE argument is used to define the initial size of the database. The default size is 1MB. The default type of the size argument is MB, but in can be specified in KB, GB and TB as well.

We should set the size of the database to 1 GB with the argument, SIZE = 1GB.

We should not set a MAXSIZE. If max_size is not specified, the file grows until the disk is full. There is not any need to set the FILESIZE argument, since the database size is not likely to grow.

Incorrect answers:

A: The SIZE=10 option sets the initial database size to 10MB. This is much too small and the database would have to grow multiple times by 10% to reach the estimated size of 1GB. This would decrease performance.

B: The SIZE=10 option sets the initial database size to 10MB. This is much too small and the database would have to grow multiple times by 5MB to reach the estimated size of 1GB. This would decrease performance.

Setting the MAXSIZE is not a good idea. The database is *about* 1GB, maybe slightly over.

C: The SIZE=100 option sets the initial database size to 100MB. This is much too small and the database would have to grow multiple times to reach the estimated size of 1GB. This would decrease performance.

Q. 74

You are a database developer for Woodgrove Bank. The company stores its sales data in a SQL Server 2000 database. You want to create an indexed view in this database. To accomplish his, you execute the script shown in the exhibit.

```
Set NUMERIC_ROUNDABORT OFF
GO
CREATE VIEW Sales
AS
 SELECT SUM(UnitPrice*Quantity*(1.00-Discount))AS Revenue,
    OrderDate, ProductID, COUNT_BIG(*) AS COUNT
  FROM dbo.OrderDetails AS OD JOIN dbo.Orders AS O
     ON OD.OrderID = O.OrderID
  GROUP BY O.OrderDate, OD.ProductID
GO
CREATE UNIQUE CLUSTERED INDEX IX_Sales ON Sales (OrderDate,
ProductID)
GO
```

The index creation fails, and you receive an error message. You want to eliminate the error message and create the index.

What should you do?

A. Add an ORDER BY clause to the view.

B. Add a HAVING clause to the view.

C. Change the NUMERIC_ROUNDABORT option to ON.

D. Change the index to a unique, nonclustered index.

E. Add the WITH SCHEMABINDING option to the view.

Answer: E.
Explanation: To create a clustered indexed on a view, the view must meet a number of requirements. One of these is that the view must be created with the SCHEMABINDING option. SCHEMABINDING binds the view to the schema of the underlying base tables.

Reference: BOL, CREATE VIEW

Incorrect answers:
A: Although a view in which the SELECT statement that defines the view can use the ORDER BY clause, the ORDER BY clause is not required in creating the clustered index.

B: If GROUP BY is specified in a view, the SELECT list must contain a COUNT_BIG(*) expression, and the view definition cannot specify the HAVING, CUBE, or ROLLUP clauses.

C: The NUMERIC_ROUNDABORT option would not prevent the creation of indexes on the view.

Note: When NUMERIC_ROUNDABORT is set to ON, an error is generated when loss of precision occurs in an expression. When it is set to OFF, losses of precision do not generate error messages and the result is rounded to the precision of the column or variable storing the result. SET NUMERIC_ROUNDABORT should be set to OFF when indexes are created or manipulated on computed columns or indexed views to keep the highest possible precision.

D: A clustered index determines the physical order of data in a table while a nonclustered index does not. A unique index on a view can be either clustered or nonclustered. A clustered index gives the best performance. We should therefore try to create a clustered index.

Q. 75
You are a database developer for your company's SQL Server 2000 database. Another database developer named Andrea needs to be able to alter several existing views in the database. However, you

want to prevent her form viewing or changing any of the data in the tables. Currently, Andrea belongs only to the Public database role.

What should you do?

A. Add Andrea to the **db_owner** database role.

B. Add Andrea to the **db_ddladmin** database role

C. Grant Andrea CREATE VIEW permissions.

D. Grant Andrea ALTER VIEW permissions.

E. Grant Andrea REFERENCES permissions on the tables.

Answer: B.
Explanation: Members of the db_ddladmin fixed database role can add, modify, or drop objects in the database. db_ddladmin can issue commands in the data definition language (DDL), but cannot issue GRANT, REVOKE, or DENY statements (statements in the Data Control Language, or DCL). db_ddladmin cannot use commands in the Data Manipulation Language (DML), such as SELECT, UPDATE, and DELETE. In short, db_ddladmin cannot view or change any data. Andrea would be able to modify views (and create and drop them), but she would not be able to change the data in tables.

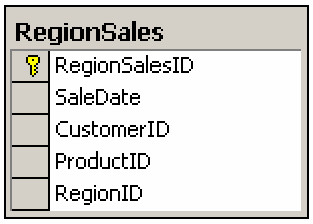
Incorrect answers:
A: Adding Andrea to the db_owner database role would give her all permissions on the table including DELETE, INSERT, and UPDATE. Andrea would be able to modify the data in the tables.

C: Andrea does not need to create views, as she needs to alter existing views. Also she must be prevented from viewing data in the table. Hence CREATE VIEW is not appropriate.

D: ALTER VIEW permissions cannot be granted.
GRANT only applies to:
CREATE DATABASE
CREATE DEFAULT
CREATE FUNCTION
CREATE PROCEDURE
CREATE RULE
CREATE TABLE
CREATE VIEW
BACKUP DATABASE
BACKUP LOG

E: The REFERENCES permission lets the owner of another table use columns in your table as the target of a REFERENCES FOREIGN KEY constraint from their table. The references permission would not give Andrea permission to alter views.

Q. 76
You are a database developer for a rapidly growing company. The company is expanding into new sales regions each month. As each new sales region is added, one or more sales associates are assigned to the new region. Sales data is inserted into a table named RegionSales, which is located in the Corporate database. The RegionSales table is shown in the exhibit.



Each sales associate should be able to view and modify only the information in the RegionSales table that pertains to his or her regions. It must be as easy as possible to extend the solution as new regions and sales associates are added.

What should you do?

A. Use GRANT, REVOKE and DENY statements to assign permission to the sales associates.

B. Use SQL Server Enterprise Manager to assign permission on the **RegionSales** table.

C. Create one view on the **RegionSales** table for each sales region.
Grant the sales associates permission to access the views that correspond to the sales region to which they have been assigned.

D. Create a new table named **Security** to hold combinations of sales associates and sales regions.
Create stored procedures that allow or disallow modifications of the data in the **RegionSales** table by validating the user of the procedures against the **Security** table.
Grant EXECUTE permissions on the stored procedures to all sales associates.

E. Create a new table named **Security** to hold combinations of sales associates and sales regions.
Create user-defined functions that allow or disallow modifications of the data.
In the **RegionSales** table by validating the user of the function against the **Security** table.
Grant EXECUTE permissions on the functions to all sales associates.

Answer: D.
Explanation:
This solution is somewhat complex but the only one that actually works. It requires the creation of a new table that holds FOREIGN KEY constraints on the RegionSales table. We would create a stored procedure with a department parameter. The procedure would look something like:

```
CREATE PROCEDURE RegionalSalesProcedure (@Region as char(50))
AS
IF  %Check against the new security table%
THEN
BEGIN
        SELECT RegionalSalesID, SaleDate, CustomerID, ProductID, RegionID
        FROM RegionSales
        WHERE RegionID=@Region
END
```

We then grant EXECUTE permission to the procedure to the Sales associates.

Note: A stored procedure is a group of Transact-SQL statements compiled into a single execution plan that can return data as OUTPUT parameters, which can return either data such as an integer or character value or a cursor variable; as return 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 and can also improve performance as they contain many tasks that would otherwise be implemented as a series of SQL statements.

Incorrect answers:

A: Although granting permissions to the specific columns that the particular sales associates require access to, and denying the permissions to the others, would allow us to specify which sales associates could view and modify data in which columns, this solution is cumbersome; we would have to determine whether users have access to the tables by way of membership in other roles and whether users have permissions at higher levels of the database.

B: SQL Server Enterprise Manager is the primary administrative tool for SQL Server 2000 and provides a Microsoft Management Console (MMC) compliant user interface for monitoring and implementing administrative tasks. Although this can be used to assign permissions, it is awkward, as we would have to determine whether users have access to the tables by way of membership in other roles and whether users have permissions at higher levels of the database.

C: Just creating a view and assigning permission to the view will not work. One additional table would be required.

Note: *A possible solution:*

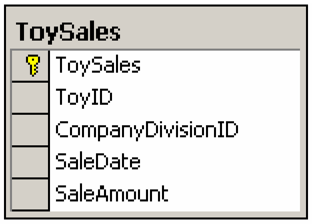
1) Create a new table named Security to hold combinations of sales associates and sales regions.
2) Create a view to get data needed from RegionSales join Security by RegionID, and use WITH CHECK OPTION in the view for WHERE Sales_Associate=USER.

E: This proposed solution is very similar to the correct answer. Instead of a stored procedure, we use a user-defined function. User-defined functions are either scalar-valued or table-valued. We must use a table-valued user-defined function since we must produce a row set. Table-valued user-defined functions populate a table return variable. A solution with a user-defined function seems like a possible solution. There are however more restrictions to user-defined functions compared to stored procedures. Therefore this is not the preferred solution.

Reference: BOL, Create Function.

Q. 77

You are a database developer for a toy company. Another developer, Marie, has created a table named ToySales. Neither you nor Marie is a member of the sysadmin fixed server role, but you are both members of the db_owner database role. The ToySales table stores the sales information for all departments in the company. This table is shown in the exhibit.



You have created a view under your database login named vwDollSales to display only the information from the ToySales table that pertains to sales of dolls. Employees in the dolls department should be given full access to the data. You have also created a view named vwActionFigureSales to display only the information that pertains to sales of action figures. Employees in the action figures department should be given full access each other's data. The two departments currently have no access to the data.

Employees in the data department are associated with the Doll database role. Employees in the action figures department are associated with the ActionFigure database role.

You must ensure that the two departments can view only their own data. Which three actions should you take? (Each correct answer presents part of the solution. Choose three)

A. Transfer the ownership of the table and the views to the database owner.

B. Grant SELECT permissions on the **ToySales** table to your login.

C. Grant SELECT permissions on the **vwDollSales** view to the **Doll** database role.

D. Grant SELECT permission on the **vwActionFigureSales** view to the **ActionFigure** database role.

E. Deny SELECT permission on the **ToySales** table for the **Doll** database role.

F. Deny SELECT permissions on the **ToySales** table for the **ActionFigure** database role.

Answer: A, C, D.
Explanation:
A: In order to avoid a broken ownership chain, the ownership of the table must be transferred from Maria to the database owner. You, as a member of the db_owner database role, would then be the owner of both the view and the table.

C: The **Doll** database role requires SELECT permission to the **vwDollSales** view

D: The **ActionFigure** database role requires SELECT permission to the **vwActionFigureSales** view.

Reference: BOL, Using Ownership Chains

Incorrect answers:
B: As a member of the db_owner role, you already have access to all objects in the database. Assigning grant permissions on the ToySales table to your login is thus not required.

E: We should grant permission, not deny permission. We should grant permission on the view, not on the table.

Note: Deny permissions override all other permissions at the level on which the deny permission was specified and the levels below that. Thus the deny permissions applied at the level of the base table will be in effect on the view that is based on that table.

F We should grant permission, not deny permission. We should grant permission on the view, not on the table.

Q. 78
You are a database developer for your company's SQL Server 2000 database. The database is installed on a Microsoft Windows 2000 Server computer. The database is in the default configuration. All tables in the database have at least one index. SQL Server is the only application running on the server.

The database activity peaks during the day, when sales representatives enter and update sales transactions. Batch reporting is performed after business hours. The sales representatives report slow updates and inserts.

What should you do?

A. Run system monitor on the **SQL Server:Access Methods** counter during the day. Use the output from System Monitor to identify which tables need indexes.

B. Use the **sp_configure** system stored procedure to increase the number of locks that can be used by SQL Server.

C. Run SQL Profiler during the day. Select the **SQL:BatchCompleted** and **RPC:Completed** events and the **EventClass** and **TextData** data columns. Use the output from SQL profiler as input to the Index Tuning Wizard.

D. Increase the value of the **min server memory** option.

E. Rebuild indexes, and use a FILLFACTOR of 100.

Answer: C.
Explanation: The Index Tuning Wizard can be used to analyze queries and determine which indexes should be created on a table, and 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.

Incorrect answers:
A: SQL Server provides objects and counters that can be used by System Monitor in a Windows 2000 or by Performance Monitor in Windows NT 4.0 to monitor activity on computers running an instance of SQL Server. The Access Methods object SQL Server provides counters to monitor how the logical pages within the database are accessed. Monitoring the methods used to access database pages can help in determining whether query performance can be improved by adding or modifying indexes or by rewriting queries.

B: sp_configure can be used to set the maximum number of available locks on a SQL Server, thereby limiting the amount of memory SQL Server 2000 uses for locks. The default setting is 0, which allows SQL Server to allocate and deallocate locks dynamically based on changing system requirements. When the server is started with locks set to 0, the lock manager allocates two percent of the memory allocated to SQL Server to an initial pool of lock structures. As the pool of locks is exhausted, additional locks are allocated. The dynamic lock pool does not allocate more than 40 percent of the memory allocated to SQL Server.

D: The min server memory and max server memory configuration options specify upper and lower limits to the amount of memory used by the SQL Server database engine. The database engine does not immediately acquire the amount of memory specified in min server memory. The database engine starts with only the memory required to initialize. As the database engine workload increases, it keeps acquiring the memory required to support the workload and will not free any of the acquired memory until it reaches the amount specified in min server memory.

E: FILLFACTOR accommodates future expansion of table data and can reduce possible defragmentation of the table. FILLFACTOR specifies the percentage of the index page to be left empty. It is recommended that a FILLFACTOR of 100 be used only if no INSERT or UPDATE statements will occur, such as with a read-only table.

Q. 79
You are a database developer for a shipping company. You have a SQL Server 2000 database that stores order information. The database contains tables named Order and OrderDetails. The database resides on a computer that has four 9-GB disk drives available for data storage. The computer has two disk controllers. Each disk controller controls two of the drives. The Order and OrderDetail tables are often joined in queries.

You need to tune the performance of the database. What should you do? (Each correct answer presents part of the solution. Choose two.)

A. Create a new filegroup on each of the four disk drives.

B. Create the clustered index for the **Order** table on a separate filegroup from the nonclustered indexes.

C. Store the data and the clustered index for the **OrderDetail** table on one filegroup, and create the nonclustered indexes on another filegroup.

D. Create the **Order** table and its indexes on one filegroup, and create the **OrderDetail** table and its indexes on another filegroup.

E. Create two filegroups that each consists of two disk drives connected to the same controller.

Answer: D, E.
Explanation: We need to increase performance, and we will concentrate on making the joins between the **Order** and the **OrderDetail** table optimized, since these joins occur frequently.

We have to make two decisions:
∉ how will the filegroups be created
∉ how will the data and indexes be distributed on the filegroups

Filegroups
A filegroup is a group of files. We don't have to put a filegroup on every disk. Instead we can use two filegroups, that each consists of two disk drivers connected to the same controller. This will make administration easier. (Answer E.)

Data and indexes
The tables are frequently joined. We should therefore separate them to increase performance. (Answer D.)

Incorrect answers:
A: Creating a filegroup in each drive is not a bad idea in general. The point is that performance will improve a separate thread is created for each file on each disk array in order to read the table's data in parallel.

Here, however, we use the fact that there are two different disk controllers and that we use many joins to find another better solution.

B: There is only one index of concern in the Order table and it is the clustered index. It is not crucial for the performance of the joins to separate the clustered and the non-clustered indexes of the Orders table.

C: Separating the clustered index and the nonclustered indexes is a common practice to improve performance. It is not the best solution in this scenario, however. It would be the next step in improving performance further.

Q. 80
You are the database developer for a brokerage firm. The database contains a table named Trades. The script that was used to create this table is shown in the Script for Trades Table exhibit.

```
CREATE TABLE Trades
      (
      TradeID int IDENTITY(1,1)PRIMARY KEY NONCLUSTERED NOT NULL,
      TradeDate datetime NULL,
      SettlementDate datetime NULL,
      Units decimal(18, 4) NULL,
      Symbol char(5) NULL,
      ClientID int NULL,
      BrokerID int NULL
      )
GO
CREATE CLUSTERED INDEX c_symbol ON Trades (Symbol)
```

The Trades table has frequent inserts and updates during the day. Reports are run against the table each night. You execute the following statement in the SQL Query Analyzer:

DBCC SHOWCONTIG (Trades)

The output for the statement is shown in the DBCC Statement Output exhibit.

DBCC Statement Output

```
DBCC SHOWCONTIG scanning 'Trades' table. . . . .
Table: 'Trades'(1621580815); index ID:1, database ID:12
Table level scan performed.
-Pages Scanned-----------------------------------------:104
-Extents Scanned---------------------------------------:16
-Extent Switches----------------------------------------:24
-Avg. Pages per Extent--------------------------------:6.5
-Scan Density[Best Count:Actual Count]-----------:52.00%[13:25]
-Logical  Scan Fragmentation-------------------------:7.69%
-Extent Scan Fragmentation---------------------------:43.75%
-Avg. Bytes Free per page-----------------------------:460.1
-Avg. Page Density (full)------------------------------:94.32%
DBCC execution completed. If DBCC printed error messages, contact
your system
```

You want to ensure optional performance for the insert and select operations on the Trades table. What should you do?

A. Execute the DBCC DBREINDEX statement on the table.

B. Execute the UPDATE STATISTICS statement on the table.

C. Execute the DROP STATISTICS statement on the clustered index.

D. Execute the DBCC INDEXDEFRAG statement on the primary key index.

E. Execute the DROP INDEX and CREATE INDEX statements on the primary key index.

Answer: A.
Explanation: In our scenario we see that Logical Scan Fragmentation is 10% and acceptable, but the Extent Scan Fragmentation is 43.75% and is a clear indication that the table is fragmented. To solve the fragmentation problem, the index should be rebuilt.

Note 1: In general, you need to do very little in terms of index maintenance. However, indexes can become fragmented. You can use the DBCC SHOWCONTIG command to report on the fragmentation of an index. You can use either the Extent Scan Fragmentation value or the Logical Scan Fragmentation value to determine the general level of fragmentation in a table. Logical Scan Fragmentation and, to a lesser extent, Extent Scan Fragmentation values give the best indication of a table's fragmentation level. Both these values should be as close to zero as possible (although a value from 0% through 10% may be acceptable).

Note 2: DBCC DBREINDEX rebuilds an index for a table or all indexes defined for a table. By allowing an index to be rebuilt dynamically, indexes enforcing either PRIMARY KEY or UNIQUE constraints can be rebuilt without having to drop and re-create those constraints. This means an index can be rebuilt without knowing the table's structure or constraints, which could occur after a bulk copy of data into the table. If either index_name or fillfactor is specified, all preceding parameters must also be specified. DBCC DBREINDEX can rebuild all of the indexes for a table in one statement, which is easier than coding multiple DROP INDEX and CREATE INDEX statements. Because the work is done by one statement, DBCC DBREINDEX is automatically atomic, while individual DROP INDEX and CREATE INDEX statements would have to be put in a transaction to be atomic. Also, DBCC DBREINDEX can take advantage of more optimizations with DBCC DBREINDEX than it can with individual DROP INDEX and CREATE INDEX statements. DBCC DBREINDEX is not supported for use on system tables.

Incorrect answers:
B: Updating statistics could improve performance, but the index of the table is fragmented, so it would be a better idea to rebuild the index.

Note: SQL Server keeps statistics about the distribution of the key values in each index and uses these statistics to determine which index(es) to use in query processing. Query optimization depends on the accuracy of the distribution steps. UPDATE STATISTICS should be run if there is a significant change in the key values in the index, or if a large amount of data in an indexed column has been added, changed, or removed, or if the table has been truncated using the TRUNCATE TABLE statement.

C: Dropping the statistics of the table would not improve query performance. On the contrary, it could decrease performance.

D: Recreating a single index is not the best solution, since the DBCC DBREINDEX command rebuilds all the indexes automatically. It is better to rebuild all the indexes, not just a single index.

Note: DBCC INDEXDEFRAG defragments clustered and secondary indexes of the specified table or view. DBCC INDEXDEFRAG can defragment clustered and nonclustered indexes on tables and views. DBCC INDEXDEFRAG defragments the leaf level of an index so that the physical order of the pages matches the left-to-right logical order of the leaf nodes, thus improving index-scanning performance. Unlike DBCC DBREINDEX (or the index-building operations in general), DBCC INDEXDEFRAG is an online operation. It does not hold locks long term and thus will not block running queries or updates. In addition, the defragmentation is always fully logged, regardless of the database recovery model setting (see ALTER DATABASE). Also, DBCC INDEXDEFRAG will not help if two indexes are interleaved on the disk because INDEXDEFRAG shuffles the pages in place. To improve the clustering of pages, rebuild the index.

E Dropping and recreating a single index is not the best solution, since the DBCC DBREINDEX command rebuilds all the indexes automatically. It is better to rebuild all the indexes, not just a single index.

Q. 81
You are the developer of a database named Inventory. You have a list of reports that you must create. These reports will be run at the same time.

You write queries to create each report. Based on the queries, you design and create the indexes for the database tables.

You want to ensure that you have created useful indexes. What should you do?

A. Create a SQL Profiler trace, and use the **Objects** event classes.

B. Run the Index Tuning Wizard against a workload file that contains the queries used in the reports.

C. Run System Monitor, and use the **SQLServer:Access Methods** counter.

D. Execute the queries against the tables in SQL Query Analyzer, and use the SHOWPLAN_TEXT option.

Answer: B.
Explanation: In this scenario, the reports have been created, as have the indexes based on them. We must ensure that we have created useful indexes. The queries will be run simultaneously so we are better off examining their performance with the Index Tuning Wizard than by examining them one-by-one with SQL Query Analyzer.

The Index Tuning Wizard is usually used to analyze and tune the indexes of the database. This is done by running it on a workload. The SQL Profiler is used to create a trace file that is used by the Index Tuning Wizard. In our case, the SQL Profiler would be used to create a trace, which would include report queries running simultaneously.

You can also use the wizard purely for analyzing your existing design. One of the reports, Index Usage Report (Current Configuration), shows what percentage of the queries in the submitted workload make use of each of your existing indexes. You can use this information to determine whether an index is being used at all and get rid of any unused indexes along with their space and maintenance overhead.

Incorrect Answers:

A: Monitoring object events in SQL profiler would not be the best way to determine if useful indexes have been created.

Note: SQL Profiler is a graphical tool that allows system administrators to monitor events in an instance of SQL Server 2000. It can be used to capture and save data about each event to a file or SQL Server table for later analysis and can be filtered to monitor only the events of interest. Monitoring too many events adds overhead to the server and the monitoring process, and can cause the trace file or trace table to grow very large, especially when the monitoring process takes place over a long period of time. SQL Profiler can be used to monitor the performance of an instance of SQL Server, to debug Transact-SQL statements and stored procedures, and to identify slow-executing queries.

C: The System Monitor cannot directly be used to ensure the usefulness of SQL Server indexes. System Monitor can produce performance statistics and raise alerts when predefined counter thresholds are met.

Note: SQL Server provides objects and counters that can be used by System Monitor in Windows 2000, or by Performance Monitor in Windows NT 4.0, to monitor activity on computers running an instance of SQL Server. The Access Methods object in SQL Server provides counters to monitor how the logical pages within the database are accessed. Monitoring the methods used to access database pages can help in determining whether query performance can be improved by adding or modifying indexes or by rewriting queries.

D: The SET SHOWPLAN_ALL command causes SQL Server not to execute Transact-SQL statements. Instead, SQL Server returns detailed information about how the statements are executed and provides estimates of the resource requirements for the statements. Howeve,r this method has two main drawbacks:

Á The output is textual, which makes it hard to read.
Á It doesn't take into account that the queries are run simultaneously. You would get one report for each query, not a global assessment like the Index Tuning Wizard provides.

Note: SQL Query Analyzer is a graphical user interface that can be used to design and test Transact-SQL statements, batches, and scripts interactively. SQL Query Analyzer can be called from SQL Server Enterprise Manager. In SQL Server 2000 the Transact-SQL SET statement options STATISTICS TIME and STATISTICS IO are used to retrieve information that could aid in diagnosing long-running queries.

Q. 82
You are a database developer for your company's SQL server 2000 database. You use the following script to create a view named Employee in the database:

```
CREATE VIEW Employee AS
SELECT P.SSN, P.LastName, P.FirstName, P.Address, P.City, P.State,
       P.Birthdate, E.EmployeeID, E.Department, E.Salary
FROM Person AS P JOIN Employees AS E ON (P.SSN = E.SSN)
```

The view will be used by an application that inserts records in both the Person and Employees tables. The script that was used to create these tables is shown in the exhibit.

```
CREATE TABLE Person
(
SSN char(11) NOT NULL PRIMARY KEY
LastName varchar (50) NOT NULL,
FirstName varchar (50) NOT NULL,
Address varchar (100) NOT NULL,
City varchar (50) NOT NULL,
State char (2) NOT NULL,
Birthdate datetime NOT NULL
)
GO
CREATE TABLE Employees
(
EmployeeID int NOT NULL PRIMARY KEY,
SSN char (11) UNIQUE NOT NULL,
Department varchar (10) NOT NULL,
Salary money NOT NULL,
CONSTRAINT FKEmpPER FOREIGN KEY (SSN)REFERENCES Person (SSN)
)
```

You want to enable the application to issue INSERT statements against the view. What should you do?

A. Create an AFTER trigger on the view.

B. Create an INSTEAD OF trigger on the view.

C. Create an INSTEAD OF trigger on the Person and Employees tables.

D. Alter the view to include the WITH CHECK option.

E. Alter the view to include the WITH SCHEMABINDING option.

Answer: B.
Explanation: INSTEAD OF triggers override the normal actions of the statement that fires the trigger. They can be defined on either tables or views. INSTEAD OF triggers are most useful for extending the types of updates a view can support and can provide the logic to modify multiple base tables through a view.

Incorrect answers:
A: AFTER trigger can only be created on tables and not on views. INSTEAD of triggers can however be created on views.

C: INSTEAD OF triggers override the normal actions of the statement that fires the trigger. They can be defined on either tables or views. INSTEAD OF triggers are most useful for extending the types of updates a view can support and can provide the logic to modify multiple base tables through a view. However, the application will access and work on the view and not on the underlying tables. Hence, placing a trigger on the underlying tables will not be fired by statements in the view and the application will be unable to effect changes in the tables.

D: A view cannot make updates in two base tables. The WITH CHECK OPTION would not help.

Note: The WITH CHECK OPTION clause forces all data modification statements executed against a view to adhere to the criteria set in the SELECT statement used to create or define the view. Hence, rows cannot be modified in a way that causes them to fall outside of the scope of the view or cause them to disappear from the view. Any modification that would cause this to happen is cancelled and an error is displayed.

E: Views or tables participating in a view created with the SCHEMABINDING clause cannot be dropped, unless the view is dropped or changed so that it no longer has schema binding. In addition, ALTER TABLE statements on tables that participate in views having schema binding will fail if these statements affect the view definition.

Q. 83
You are a database developer for Wide World Importers. You are creating a table named Orders for the company's SQL Server 2000 database. Each order contains an OrderID, an OrderDate, a CustomerID, a ShipperID, and a ShipDate.

Customer services representatives who take the orders must enter the OrderDate, CustomerID, and ShipperID when the order is taken. The OrderID must be generated automatically by the database and must be unique. Orders can be taken from existing customers only. Shippers can be selected only from an existing set of shippers. After the customer service representatives complete the order, the order is sent to the shipping department for final processing. The shipping department enters the ship date when the order is shipped.

Which script should you use to create the Orders table?

A.
```
CREATE TABLE Orders
(
OrderID uniqueidentifier PRIMARY KEY NOT NULL,
OrderDate datetime NULL,
CustomerID char(5) NOT NULL FOREIGN KEY REFERENCES Customer (Customer ID),
ShipperID int NOT NULL FOREIGN KEY REFERENCES Shippers(ShipperID),
ShipDate datetime Null
)
```

B.
```
CREATE TABLE Orders
(
OrderID int identity (1, 1) PRIMARY KEY NOT NULL,
OrderDate datetime NOT NULL,
CustomerID char(5) NOT NULL FOREIGN KEY REFERENCES Customer (Customer ID),
ShipperID int NOT NULL FOREIGN KEY REFERENCES Shippers(ShipperID),
ShipDate datetime Null
)
```

C.
```
CREATE TABLE Orders
(
OrderID int identity (1, 1) PRIMARY KEY NOT NULL,
OrderDate datetime NULL,
CustomerID char(5) NOT NULL FOREIGN KEY REFERENCES Customer (Customer ID),
ShipperID int NULL,
ShipDate datetime Null
)
```

D.
```
CREATE TABLE Orders
(
OrderID uniqueidentifier PRIMARY KEY NOT NULL,
OrderDate datetime NOT NULL,
CustomerID char(5) NOT NULL FOREIGN KEY REFERENCES Customer (Customer ID),
ShipperID int NOT NULL FOREIGN KEY REFERENCES Shippers(ShipperID),
ShipDate datetime Null
)
```

Answer: B.
Explanation: In this scenario, the rows must be identified from the OrderID column, hence OrderID must be designated with the **identity** property. Furthermore, the value inserted into the OrderID column must be generated automatically and must be unique. This can be accomplished by assigning a seed, which is the value that is used for this column in the first row of the table, and an increment, which is the value that must be added to the identity value of the previous row that was loaded. This is expressed in the form **identity** **(***seed*, *increment***)**. Either both the seed and increment must be specified, or neither must be specified. The ShipDate column must allow nulls as it is to be filled in when the order is shipped, hence when the order is completed. In addition, as the OrderID column will be searched in order to enter the ShipDate value at a later stage, the OrderID must be the FOREIGN KEY. Because orders can only be taken from existing customers and shippers must be existing shippers, FOREIGN KEY constraints must exist against the CustomerID in the Customer table and against the ShipperID column in the Shippers table.

Incorrect answers:
A: The uniquely identified OrderID column in this script will not be generated automatically as it does not contain a seed and an increment in its definition.

C: The code in this solution does not ensure referential integrity between the ShipperID column in the Orders table and the ShipperID column in the Shippers table. Referential integrity is ensured through the use of FOREIGN KEY constraints and is required to ensure that orders are only taken from existing customers and toensure that shippers are selected only from the existing set of shippers.

D: The uniquely identified OrderID column in this script will not be generated automatically as it does not contain a seed and an increment in its definition.

Q. 84
You are a database developer for Lucerne Publishing. The company stores its sales data in a SQL Server 2000 database. This database contains a table named Orders. There is currently a clustered index on the table, which is generated by using a customer's name and the current date.

The orders table currently contains 750,000 rows, and the number of rows increased by 5 percent each week. The company plans to launch a promotion next week that will increase the volume of inserts to the Orders table by 50 percent.

You want to optimize inserts to the Orders table during the promotion. What should you do?

A. Create a job that rebuilds the clustered index each night by using the default FILLFACTOR.

B. Add additional indexes to the **Orders** table.

C. Partition the **Orders** table vertically.

D. Rebuild the clustered index with a FILLFACTOR of 50.

E. Execute the UPDATE STATISTICS statement on the **Orders** table.

Answer: D.
Explanation: The table's size will increase with 50% during the next week. The database would need to grow and index pages would have to be split. This will degrade performance. It is better to anticipate this growth by rebuilding the data pages, and by rebuilding the clustered index. It is also import to choose an appropriate FILLFACTOR. The default value is 80. By choosing a FILLFACTOR of 50, the pages would be around 75% filled after the promotion and would be allow to grow further.

Note: FILLFACTOR
It is seldom necessary to specify a fill factor when you create an index. The option is provided for fine-tuning performance. It is useful when you are creating a new index on a table with existing data, and particularly when you can accurately predict future changes in that data.

FILLFACTOR is specified as a percentage, and it indicates how full SQL Server should make the leaf level of each index page during index creation. As a table is updated and the index page fills, SQL Server must split the index page to make room for new rows, which is requires time. For update-intensive tables, a properly chosen FILLFACTOR value yields better update performance than an improper FILLFACTOR value. The default FILLFACTOR is usually efficient for most database usage. An explicit FILLFACTOR setting applies only when the index is first created. SQL Server does not dynamically keep the specified percentage of empty space in the pages. The default fill factor is 80.

Incorrect answers:
A: Rebuilding the clustered index every night with the default FILLFACTOR requires more administration. The default FILLFACTOR is 80, and if we assume a few weeks during which the 50% increase takes place, we see that the data pages would fill up completely, especially the first day. This would increase index splitting and decrease performance.

B: Adding the correct indexes to a table can be useful for improving the performance of query on a table. Adding indexes, however, could hamper INSERT operations, since they would be added to existing index pages and would reduce the amount of space left in those pages for the indexes of new rows inserted into the table.

Instead, we should anticipate the 50% growth by rebuilding the indexes with a lower fillfactor.

C: Partitions are used to store and manage pre-calculated aggregations and, sometimes, source data. They also offer flexibility in storing such data in multiple locations and in optimizing its access. This would not, however, anticipate the 50% increase of rows in the table.

E: UPDATE STATISTICS should be re-run when there is significant change in the key values in the index such as occurs when a large amount of data in an indexed column has been added, changed, or removed, i.e., if the distribution of key values has changed, or the table has been truncated using the TRUNCATE TABLE statement and then repopulated. In this scenario you are expecting an increase in the volume of data to be inserted into the table. This has not yet occurred, therefore re-running UPDATE STATISTICS is not required. Furthermore, the UPDATE STATISTICS function is related to indexing, which is of concern when queries are run against the table. It does not improve INSERT operations.

Q. 85
You are designing your company's Sales database. The database will be used by three custom applications. Users who require access to the database are currently members of Microsoft Windows 2000 groups. Users were placed in the Windows 2000 groups according to their database access requirements. The custom applications will connect to the sales database through the sales database through application roles that exist for each application. Each application role was assigned a password.

All users should have access to the sales database only through the custom applications. No permissions have been granted in the database.

What should you do?

A. Assign appropriate permissions to each Windows 2000 group.

B. Assign appropriate permissions to each application role.

C. Assign the Windows 2000 groups to the appropriate application role.

D. Provide users with the password to the application role.

Answer: B.
Explanation: To set up an application role we must:

1. Create the application role.
2. Assign a password to the application role.
3. Configure the application to use the application role to access a SQL Server database.
4. Assign appropriate permissions to the application role in the SQL Server database.

In this scenario, steps 1, 2, and 3 have already been taken. We only need to assign appropriate permissions to the application role.

Note: Application roles contain no members, and Windows NT 4.0 or Windows 2000 groups, users, and roles cannot be added to application roles. Permissions of the application role are gained only when the application role is activated for the user's connection through a specific application or applications. In other words, a user's association with an application role is due to his or her ability to run the application. Furthermore, application roles are inactive by default and require a password to be activated, and they bypass standard permissions. Thus, when an application role is activated, i.e. when the user runs the application, all permissions applied to the user's login, user account, or as a member of a group or database role in all databases are temporarily suspended for the duration of the connection, and the permissions associated with the application role for the database in which the application role exists are brought to bear on the connection. This ensures that all the functions of the application can be performed regardless of the permissions that apply to the user. Users and Windows NT 4.0 or Windows 2000 groups do not need to be granted any permissions for the database because all permissions can be assigned by the applications they use to access the database. In such an event, a system-wide password can be assigned to an application role if access to the applications is secure.

Incorrect answers:
A: The permissions must be assigned to the application role not to the Windows group.

C: Application roles contain no members, and Windows NT 4.0 or Windows 2000 groups, users, and roles cannot be added to application roles.

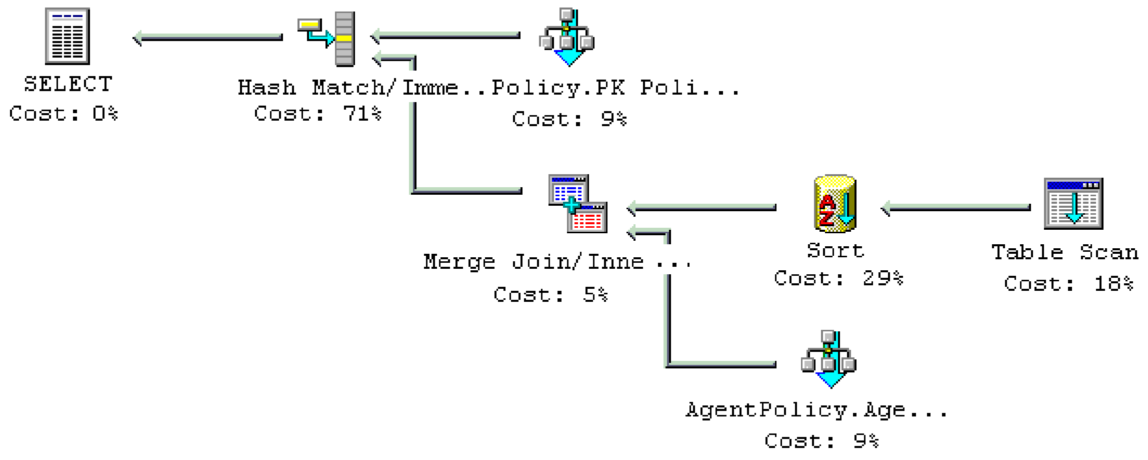
D: The application role uses a password to connect to SQL servers. The users doesn't use the application role directly, they use it through the application.

Q. 86
You are a database developer for your company's database named Insurance. You execute the following script in SQL Query Analyzer to retrieve agent and policy information:

```
SELECT A.LastName, A.FirstName, A.CompanyName, P.PolicyNumber
FROM Policy AS P JOIN AgentPolicy AS AP
ON (P.PolicyNumber = AP.PolicyNumber)
```

```
    JOIN Agents AS A ON (A.AgentID= AP.AgentID)
```

The query execution plan that is generated is shown in the Query Execution Plan exhibit:



The information received when you move the pointer over the Table Scan icon is shown in the Table Scan Information exhibit:

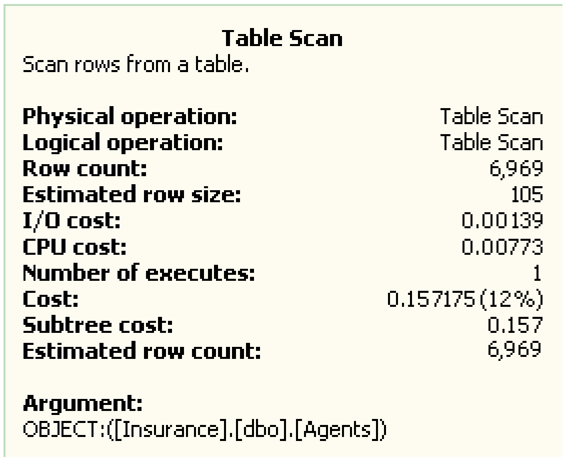


Table Scan
Scan rows from a table.

Physical operation:	Table Scan
Logical operation:	Table Scan
Row count:	6,969
Estimated row size:	105
I/O cost:	0.00139
CPU cost:	0.00773
Number of executes:	1
Cost:	0.157175(12%)
Subtree cost:	0.157
Estimated row count:	6,969

Argument:
OBJECT:([Insurance].[dbo].[Agents])

You want to improve the performance of this query. What should you do?

A. Change the order of the tables in the FROM clause to list the **Agents** table first.

B. Use a HASH JOIN hint to join the **Agents** table in the query.

C. Create a clustered index on the **AgentID** column of the **Agents** table.

D. Create a nonclustered index on the **AgentID** column of the **Agents** table.

Answer: C.
Explanation: By the Estimated Execution Plan, and the Table Scan Information, we see that there is a Table Scan on the Agents table. This is not efficient. The Agents table is joined with AgentPolicy table on the column AgentID. By creating a clustered index on the AgentID column of the Agents table, this join will be much more efficient. It will use a Clustered Index scan instead of a table scan.

A clustered index is more efficient than a non-clustered index. There can only be one clustered index in a table. But the perfect candidate column for a clustered index in the Agents table is the AgentID, so it seems likely that there is no clustered index on the Agents table.

Incorrect Answers:
A: The order of the arguments in the FROM clause does not affect query performance.

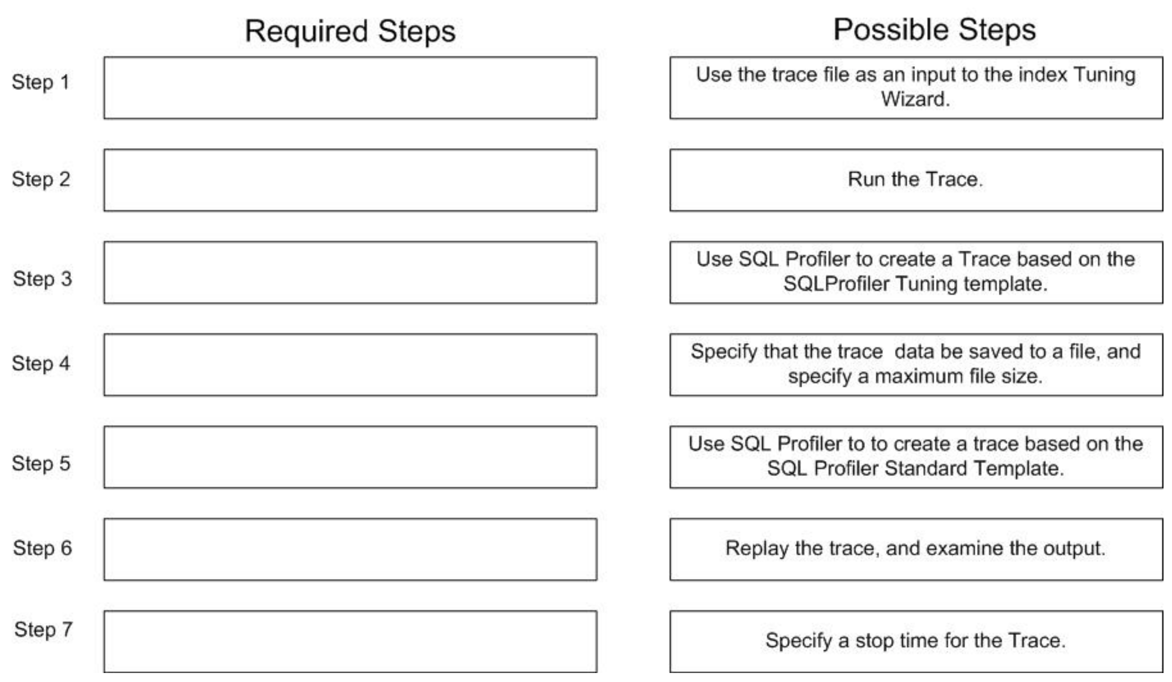
B: Hash joins are used for set-matching operations. They are similar to a merge join and can be used only if there is at least one WHERE clause in the join predicate. In this scenario the code does not contain a WHERE clause, therefore a HASH JOIN is not possible. Furthermore, the query optimizer can usually pick the best optimization method without hints being specified. However, because the query optimizer of SQL Server selects the best execution plan for a query, it is recommended that hints should only be used as a last resort by experienced developers and database administrators.

D: A clustered index is more efficient than a non-clustered index. There can only be one clustered index in a table. But the perfect candidate column for a clustered index in the Agents table is the AgentID, so it seems likely that there is no clustered index on the Agents table.

Q. 87
You are a database consultant. One of your customers reports slow query response times for a SQL Server 2000 database, particularly when table joins are required.

Which steps should you take to analyze this performance problem?

To answer, click the Select and Place button, and then drag the appropriate transaction order choices beside the appropriate transaction steps. (Use only order choices that apply.)



	Required Steps	Possible Steps
Step 1		Use the trace file as an input to the index Tuning Wizard.
Step 2		Run the Trace.
Step 3		Use SQL Profiler to create a Trace based on the SQLProfiler Tuning template.
Step 4		Specify that the trace data be saved to a file, and specify a maximum file size.
Step 5		Use SQL Profiler to to create a trace based on the SQL Profiler Standard Template.
Step 6		Replay the trace, and examine the output.
Step 7		Specify a stop time for the Trace.

Answer:

Step 1 Use SQL Profiler to create a trace based on the SQL Profiler Tuning Template
Step 2 Specify that the trace data be saved to a file, and specify a maximum file size.
Step 3 Specify a stop time for the trace.
Step 4 Run the trace.
Step 5 Use the trace file as an input to the Index Tuning Wizard.
Step 6 Replay the trace, and examine the output.
Step 7 (none)

Explanation:
When used together, SQL Profiler and the Index Tuning Wizard provide a very powerful tool combination to help database administrators ensure that proper indexes are placed on tables and views.

The Index Tuning Wizard analyzes information from the captured workload and information about the table structures, and then presents recommendations about what indexes should be created to improve performance.

Capturing Profiler Information to Use with the Index Tuning Wizard:

Set up SQL Profiler

1. Start SQL Profiler.
2. Create a new SQL Profiler trace.
3. Select the **SQL Profiler Tuning** template from the dropdown list box.

4. Specify that the trace data be saved to a file, and specify a maximum file size.
5. Specify a stop time for the trace.
   It isn't necessary to specify a stop time. The alternative is to stop it manually later.
6. Click **Run**.

Load the trace file or table into Index Tuning Wizard

1. Start the Index Tuning Wizard.
2. Load the trace file or table into Index Tuning Wizard.
3. Run the trace file.

After the Index Tuning Wizard has completed, you can choose to update the indexes. After that, the trace file can be replayed, and hopefully the performance improvements will be evident.

Incorrect answers:
Use SQL Profiler to create a trace based on the SQL Profiler Standard template.
When the trace produced by the SQL Profiler is going to be used by the Index Tuning Wizard, the **SQL Profiler Tuning** template (not the **SQL Profiler Standard** template) should be used.

Q. 88
You are a database developer for Wingtip Toys. The company stores its sales information in a SQL Server 2000 database. This database contains a table named Orders. You want to move old data from the orders table to an archive table. Before implementing this process, you want to identify how the query optimizer will process the INSERT statement.

You execute the following script in SQL Query Analyzer:

```
SET SHOWPLAN_TEXT ON
GO
CREATE TABLE Archived_Orders_1995_1999
(
OrderID int,
CustomerID char (5),
EmployeeID int,
OrderDate datetime,
ShippedDate datetime
)

INSERT INTO Archived_Orders_1995_1999
SELECT OrderID, CustomerID, EmployeeID, OrderDate, ShippedDate
```

```
FROM SalesOrders
WHERE ShippedDate < DATEADD (year, -1, getdate())
```

You receive the following error message:

```
Invalid object name 'Archived_Orders_1995_1999'.
```

What should you do to resolve the problem?

A. Qualify the **Archived_Orders_1995_1999** table name with the owner name.

B. Request CREATE TABLE permissions.

C. Create the **Archived_Orders_1995_1999** table before you execute the SET SHOWPLAN_TEXT ON statement.

D. Change the table name to **ArchivedOrders**.

Answer: C.
Explanation: The INSERT INTO statement inserts rows into an existing table. Before SQL Query Analyzer can analyze a query, the table that the query will reference must exist, as SQL Query Analyzer tests the operation of the query against the table. In this scenario, we must create the table before the showplan can be produced.

Incorrect answers:
A: It is not necessary to qualify the owner name of the table with the table name when you are using the table. You are the owner of the table and will have owner permissions to the table.

B: As you are the database developer for your company, you should already have CREATE TABLE permissions.

D: Changing the table name to drop the numeric characters in it will not enable the use of SQL Query Analyzer, as the table must exist before SQL Query Analyzer can analyze a query against it.

Q. 89
You are a database developer for Woodgrove Bank. The company has a database that contains human resources information. You are designing transactions to support data entry into this database. Scripts for two of the transactions that you designed are shown in the exhibit.

Transaction 1	Transaction 2
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`
`UPDATE Customer`	`UPDATE CustomerPhone SET`
`SET CustomerName=@Name`	`PhoneNumber=@Phone`
`WHERE`	`WHERE CustomerID=@CustID`
`CustomerID=@CustID`	`AND PhoneType = @PType`
`UPDATE CustomerPhone`	`UPDATE CustomerAddress SET Street =`
`SET PhoneNumber=@Phone`	`@Street`
`WHERE`	`WHERE CustomerID=@CustID`
`CustomerID=@CustID`	`AND AddressType=@AType`
`AND PhoneType=@PType`	
	`UPDATE Customer SET CustomerName =`
`COMMIT TRANSACTION`	`@Name`
	`WHERE CustomerID = @CustID`
	`COMMIT TRANSACTION`

While testing these scripts, you discover that the database server occasionally detects a deadlock condition. What should you do?

A. In Transaction 2, move the UPDATE Customer statement before the UPDATE CustomerPhone statement.

B. Add the SET DEADLOCK_PRIORITY LOW statement to both transactions.

C. Add code that checks for server error 1205 to each script. If this error is encountered, restart the transaction in which it occurred.

D. Add the SET LOCK_TIMEOUT 0 statement to both transactions.

Answer: A.
Explanation: We have a deadlock. In this case a deadlock occurs after the following line of events:
1. Transaction 1 updates the Customer table.
2. Transaction 2 updates the CustomerPhone table.

At this point both the Customer and the CustomerPhone table are locked, and we are heading for disaster.

3. Transaction 1 wants to update the CustomerPhone table, but it is locked, so Transaction 1 starts to wait for the CustomerPhone table to become available.
4. Transaction 2 updates the CustomerAddress table.

5. Transaction 2 wants to update the Customer table, but it is locked, so Transaction 2 starts to wait for the Customer table to become available.

So Transaction 1 is waiting for Transaction 2 to release the CustomerPhone table, and Transaction 2 is waiting for transaction 1 to release the Customer table. A deadlock has occurred.

The solution in this and similar scenarios is to reorder the updates of tables so that both transactions update the tables in the same order, for example by moving the UPDATE Customer statement before the UPDATE CustomerPhone statement in Transaction 2.

Incorrect answers:

B: By setting both transactions' DEADLOCK_PRIORITY to low, SQL Server 2000 must again decide which transaction should be the preferred deadlock victim. This would not solve the problem at hand. We would still have the same situation.

Note: 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.

C: This proposed solution would restart all transactions that are deadlocked; it would not decrease the number of deadlocks, though.

Note: Additional code could be added to the transactions to check for server error 1205 messages. The code could be scripted to automatically rerun the transaction if this error message is detected. The server error 1205 message indicates that the transaction is involved in a deadlock. A deadlock situation arises 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 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.

All this said, it would be better to avoid deadlocks by simply reordering the update statements.

D: The LOCK_TIMEOUT setting allows an application to specify the maximum time that a statement waits on a blocked resource. When a statement has waited longer than the specified time, the blocked statement is cancelled, and error message 1222, "Lock request time-out period exceeded" is returned to the application. A transaction containing the statement is not rolled back or cancelled by SQL Server and must therefore have an error handler that can roll back the entire transaction and rerun it if needed.

Setting the LOCK_TIMOUT option to 0 would make the transactions lockout very quickly. It would not improve performance.

Q. 90
You are a database developer for a company that compiles statistics for baseball teams. These statistics are stored in a database named Statistics. The Players of each team are entered in a table named Rosters in the Statistics database. The script that was used to create the Rosters table is shown in the exhibit.

```
CREATE TABLE Rosters
(
RosterID int NOT NULL,
TeamID int NOT NULL,
FirstName char(20) NOT NULL,
LastName char(20) NOT NULL,
CONSTRAINT PK_Rosters PRIMARY KEY (RosterID),
CONSTRAINT FK_TeamRoster FOREIGN KEY (TeamID)
                    REFERENCES Team (TeamID)
)
```

Each baseball team can have a maximum of 24 players on the roster at any one time. You need to ensure that the number if players on the team never exceeds the maximum.

What should you do?

A. Create a trigger on the **Rosters** table that validates the data.
B. Create a rule that validates the data.
C. Create an UPDATE view that includes the WITH CHECK OPTION clause in its definition.
D. Add a CHECK constraint on the **Rosters** table to validate the data.

Answer: A.
Explanation: When inserting new rows into the Rosters table we must be able to check the number of players in this particular team. The only mechanism that can implement this is an INSERT trigger.

Note: Triggers are a special class of stored procedure defined to fire automatically when an UPDATE, INSERT, or DELETE statement is issued against a table or view. They are powerful tools that can be used to enforce business rules automatically when data is modified. Triggers can extend the integrity-checking logic of SQL Server constraints, defaults, and rules, although constraints and defaults should be used instead whenever they provide all the needed functionality. In this scenario it is a rule that must be conformed to and hence a trigger is required.

Incorrect answers:
B: Rules are used in cases were backward compatibility is required. They perform the same function as CHECK constraints. Rules cannot take the number of rows, or a subset of rows, in a table into account.

C: The WITH CHECK OPTION clause forces all data modification statements executed against the view to adhere to the criteria set within the SELECT statement through which the view was defined. This prevents rows from being modified in a way that will cause them to disappear from the view. Any modification that would cause this to happen is cancelled and an error is displayed.

The WITH CHECK OPTION would not, however, be able to take the number of specific rows in the view into account.

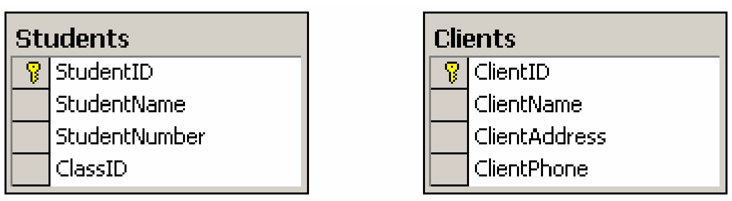
D: CHECK constraints are used to ensure data integrity and can prevent the insertion of invalid data into a column for which the inserted data type has been defined as invalid. CHECK constraints cannot take the number of rows, or a subset of rows, in a table into account.

Q. 91
You are a database developer for a consulting company. You are creating a database named Reporting. Customer names and IDs from two other databases, named Training and Consulting, must be loaded into the Reporting database.

You create a table named Customers in the Reporting database. The script that was used to create this table is shown in the Script for Customers Table exhibit.

```
CREATE TABLE Customers
(
CustomerKey int NOT NULL,
SourceID int NOT NULL,
CustomerName varchar(100) NOT NULL,
CONSTRAINT PRIMARY KEY PK_Customers (CustomerKey)
)
```

Data must be transferred into the Customers table from the Students table in the training database and from the Clients table in the Consulting database. The Students and Clients table are shown in the Students and Clients Table exhibit.



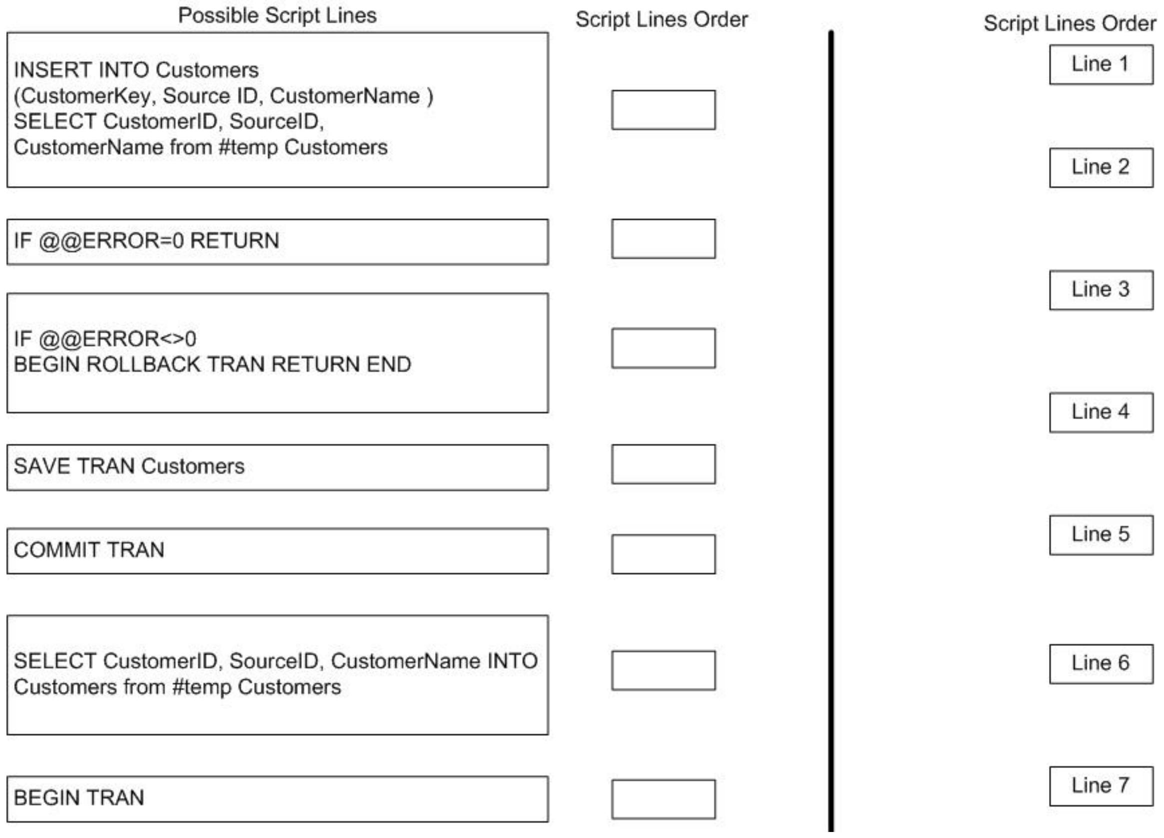
You must create a script to transfer the data into the Customers table. The first part of the script is shown in the exhibit.

Script

```
CREATE TABLE #tmpCustomers
(
CustomerID int IDENTITY(1, 1) NOT NULL,
SourceID int NOT NULL,
CustomerName varchar(100) NOT NULL,
CONSTRAINT PRIMARY KEY PK_tmpCustomers (CustomerID)
)

INSERT INTO #tmpCustomers (SourceID, CustomerName)
SELECT StudentID, StudentName FROM Training.dbo.Students

IF @@ERROR <> 0 RETURN

INSERT INTO #tmpCustomers (SourceID, CustomerName)
SELECT ClientID, ClientName FROM Consulting.dbo.Clients

IF @@ERROR <> 0 RETURN
```

To complete the script, click the Select and Place button, and then drag the appropriate line order choices beside the appropriate script lines. (Use only order choices that apply.)

SELECT AND PLACE



```
INSERT INTO Customers
(CustomerKey, Source ID, CustomerName )
SELECT CustomerID, SourceID,
CustomerName from #temp Customers
```

```
IF @@ERROR=0 RETURN
```

```
IF @@ERROR<>0
BEGIN ROLLBACK TRAN RETURN END
```

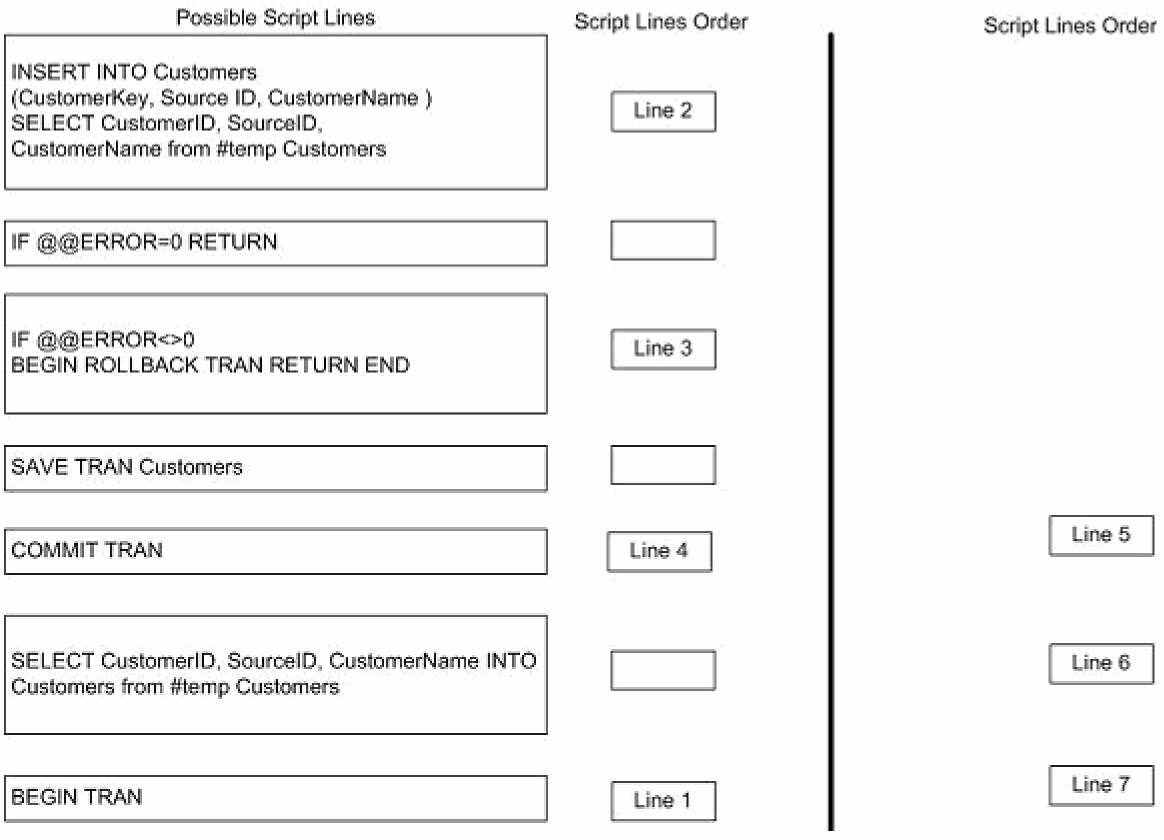
```
SAVE TRAN Customers
```

```
COMMIT TRAN
```

```
SELECT CustomerID, SourceID, CustomerName INTO
Customers from #temp Customers
```

```
BEGIN TRAN
```

Answer:



INSERT INTO Customers
(CustomerKey, Source ID, CustomerName)
SELECT CustomerID, SourceID,
CustomerName from #temp Customers
— Line 2

IF @@ERROR=0 RETURN

IF @@ERROR<>0
BEGIN ROLLBACK TRAN RETURN END
— Line 3

SAVE TRAN Customers

COMMIT TRAN — Line 4

SELECT CustomerID, SourceID, CustomerName INTO
Customers from #temp Customers

BEGIN TRAN — Line 1

Line 5
Line 6
Line 7

Explanation:

Line 1 BEGIN TRAN
Line 2 Insert INTO CustomerID
(CustomerKey, SourceID, CustomerName)
SELECT CustomerID, SourceID, CustomerName
FROM #tmpCustomers
Line 3 IF @@ERROR <> 0
BEGIN ROLLBACK TRAN
RETURN
END
Line 4 COMMIT TRAN

We start the transaction, insert the rows, check for errors and for a possible rollback of the transaction, and finally, if no errors have occurred, we commit the transaction.

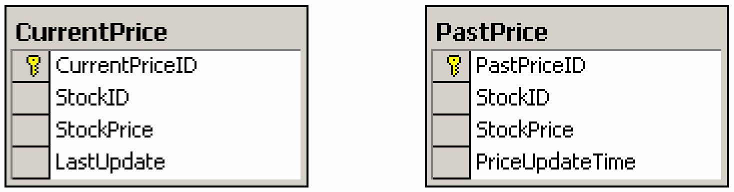
Incorrect answers:
IF @@ERROR = 0 RETURN

@@ERROR is set to 0 if the statement executed successfully and RETURN exits unconditionally from a query or procedure. RETURN is immediate and can be used at any point to exit from a procedure, batch, or statement block. Statements following RETURN are not executed. Thus the transaction is not committed.

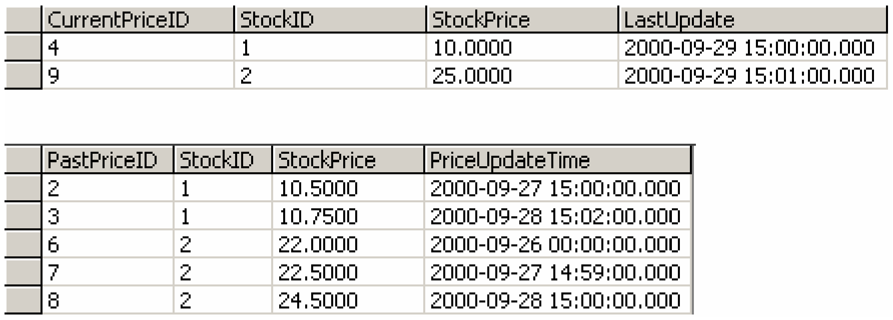
SAVE TRAN Customers
SAVE TRAN places a savepoint at a location to which a transaction can return if part of the transaction is conditionally cancelled.

SELECT CustomerID, SourceID, CustomerName
INTO Customers
FROM #tmpCustomers
Data needs to be inserted into the existing Customers table. The SELECT statement creates a new table and is thus inappropriate.

Q. 92
You are a database developer for an investment brokerage company. The company has a database named Stocks that contains tables named CurrentPrice and PastPrice. The current prices of investment stocks are stored in the CurrentPrice table. Previous stock prices are stored in the PastPrice table. These tables are shown in the CurrentPrice and PastPrice Tables exhibit.



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



CurrentPriceID	StockID	StockPrice	LastUpdate
4	1	10.0000	2000-09-29 15:00:00.000
9	2	25.0000	2000-09-29 15:01:00.000

PastPriceID	StockID	StockPrice	PriceUpdateTime
2	1	10.5000	2000-09-27 15:00:00.000
3	1	10.7500	2000-09-28 15:02:00.000
6	2	22.0000	2000-09-26 00:00:00.000
7	2	22.5000	2000-09-27 14:59:00.000
8	2	24.5000	2000-09-28 15:00:00.000

All of the rows in the CurrentPrice table are updated at the end of the business day, even if the price of the stock has not changed since the previous update. If the stock price has changed since the previous update, then a row must also be inserted into the PastPrice table.

You need to design a way for the database to execute this action automatically. What should you do?

A. Create an AFTER trigger on the **CurrentPrice** table that compares the values of the **StockPrice** column in the **inserted** and **deleted** tables. If the values are different, then the trigger will insert a row into the **PastPrice** table.

B. Create an AFTER trigger on the **CurrentPrice** table that compares the values of the **StockPrice** column in the **inserted** table with the **StockPrice** column in the **CurrentPrice** table. If the values are different, then the trigger will insert a row into the **PastPrice** table.

C. Create a cascading update constraint on the **CurrentPrice** table that updates a row in the **PastPrice** table.

D. Create a stored procedure that compares the new value of the **StockPrice** column in the **CurrentPrice** table with the old value. If the values are different, then the procedure will insert a row into the **PastPrice** table.

Answer: A.
Explanation: In this scenario it is a rule that must be conformed to. In this event a trigger is required. The trigger in this solution compares the value in StockPrice column of the **deleted** table against the value of the StockPrice column in the **CurrentPrice** table, since the **deleted** table holds the values that have been updated. When the two values differ, the trigger can insert the appropriate rows into the **PastPrice** table.

Note: The **inserted** logical table contains the recordset that has been changed (or added). The **deleted** logical table contains the original recordset as it appeared before the update.

Reference: Microsoft SQL Server 2000 Database Design and Implementation Training kit, The Inserted and Deleted Pseudo Tables, Page 350

Incorrect answers:

B: In this scenario it is a complex business rule must be conformed to. In this event a trigger is required. However, the trigger in this solution compares the value in StockPrice column of the **inserted** table against the value of the StockPrice column in the **CurrentPrice** table. As the **inserted** table holds the inserted values and as the inserts were performed against the CurrentPrice table, those values will be the same.

C: A cascading update constraint is used to enforce referential integrity and specifies that if an attempt is made to update a key value in a row that is referenced by foreign keys in existing rows in other tables, all of the foreign key values in the other tables are also updated to the new value specified for the key. In this scenario, this would mean that the Stock Price columns in the **CurrentPrice** and **PastPrice** tables are the same.

D: A trigger, not a stored procedure, is required to implement the business rule in this scenario.

Note: A stored procedure is a group of Transact-SQL statements compiled into a single execution plan that is used to return data. Stored procedures can return data as output parameters, which can return either data or a cursor variable; as return 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; or as a global cursor that can be referenced outside the stored procedure.

Q. 93

You are a database developer for Wingtip Toys. The company tracks its inventory in a SQL Server 2000 database. You have several queries and stored procedures that are executed on the database indexes to support the queries have been created.

As the number of catalogued inventory items has increased, the execution time of some of the stored procedures has increased significantly. Other queries and procedures that access the same information in the database have not experienced an increase in execution time.

You must restore the performance of the slow-running stored procedures to their original execution times. What should you do?

A. Always use the WITH RECOMPILE option to execute the slow-running stored procedures.

B. Execute the UPDATE STATISTICS statement for each of the tables accessed by the slow-running stored procedures.

C. Execute the sp_recompile system stored procedure for each of the slow-running procedures.

D. Execute the DBCC REINDEX statement for each of the tables accessed by the slow-running stored procedures.

Answer: C.
Explanation: Some queries have degraded performance. We should recreate the execution plan and the statistics on the underlying tables for these queries. This can be accomplished by recompiling them with the system stored procedure sp_recompile.

Note: As a database is changed by such actions as adding indexes or changing data in indexed columns, the original query plans used to access its tables should be optimized again by recompiling them. This optimization happens automatically the first time a stored procedure is run after SQL Server 2000 is restarted. It also occurs when an underlying table used by the stored procedure changes. The sp_recompile system stored procedure forces a recompile of a stored procedure the next time that the stored procedure is run. When the stored procedure is recompiled, the statistics on the underlying tables are recreated as well.

Incorrect answers:

A: Specifying the WITH RECOMPILE option in the stored procedure definition indicates that SQL Server should not cache a plan for this stored procedure; instead, the stored procedure is recompiled each time it is executed. The WITH RECOMPILE option should only be used when stored procedures take parameters whose values differ widely between executions of the stored procedure, resulting in different execution plans to be created each time. Use of this option causes the stored procedure to execute more slowly because the stored procedure must be recompiled each time it is executed.

B: It would be better to recompile the procedure. It would recreate the query execution plan and update the statistics on the underlying tables.

Note: SQL Server keeps statistics about the distribution of the key values in each index and uses these statistics to determine which index(es) to use in query processing. Users can create statistics on non-indexed columns by using the CREATE STATISTICS statement. However, query optimization depends on the accuracy of the distribution of the index. UPDATE STATISTICS should be re-run when there is a significant change in the key values in the index on that index as occurs when a large amount of data in an indexed column has been added, changed, or removed, i.e., if the distribution of key values has changed, or the table has been truncated using the TRUNCATE TABLE statement and then repopulated. The STATS_DATE function can be used to check when the statistics were last updated.

D: DBCC REINDEX is used to rebuild indexes. It is a time consuming process that can hold long-term locks, which can block running queries or updates. However, not all of the stored procedures' performance has deteriorated, therefore it is unlikely that indexing is responsible for some of them to have deteriorated.

Q. 94
You are a database developer for a marketing firm. You have designed a quarterly sales view. This view joins several tables and calculates aggregate information. You create a unique index on the view. You

want to provide a parameterized query to access the data contained in your indexed view. The output will be used in other SELECT lists.

How should you accomplish this goal?

A. Use an ALTER VIEW statement to add the parameter value to the view definition.

B. Create a stored procedure that accepts the parameter as input and returns a rowset with the result set.

C. Create a scalar user-defined function that accepts the parameter as input.

D. Create an inline user-defined function that accepts the parameter as input.

Answer: C.
Explanation: In this scenario we want to create a parameterized query. A parameterized query could be accomplished by a stored procedure, a scalar user-defined function, or an inline user-defined function. However, the output will be used in SELECT lists, which excludes both stored procedures and inline user-defined functions. Stored procedures or inline user-defined functions cannot be used in SELECT lists. Only scalar user-defined functions can be used in SELECT lists since they return scalar values.

Note 1: A SELECT list can contain:
- ∉ (all columns from the tables will be used)
- ∉ column_name
- ∉ expression (for example the output of a scalar user-defined function)
- ∉ IDENTITYCOL
- ∉ ROWGUIDCOL
- ∉ column_alias

Note 2: 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 scalar values, and inline user-defined functions, which return a table.

Reference:
BOL, SELECT Clause
BOL, Using the Select List

Incorrect answers:

A: Using the ALTER VIEW statement to include a parameter value in the view definition will not meet the requirements of the scenario. Only a view that produces an aggregate value can be used in a SELECT list.

B: A stored procedure cannot be used in a SELECT list.

Note: A stored procedure is a group of Transact-SQL statements that are compiled into a single execution plan. SQL Server 2000 stored procedures can return data as output parameters, which can return either data or a cursor variable; as return 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; or as a global cursor that can be referenced outside the stored procedure.

D: Inline user-defined functions, which return a table, can be used in the FROM clause, but not in the SELECT clause of a query. A table can only be used in the SELECT list if a column name is specified.

Note: It is easy to confuse the SELECT list and the FROM clause. Tables can be used in the FROM clause, but only in the SELECT list if a column name is specified as well.
For example, if we have a table Employees:
Employees.Firstname (can be used in SELECT list)
Employees (can be used in a FROM clause but not in a SELECT clause)

Q. 95
You are designing for a large grocery store chain. The partial database schema is shown in the Partial Database Schema Exhibit.

The script that was used to create the Customers table is shown in the Script for Customers Table Exhibit.

The store managers want to track customer demographics so they can target advertisements and coupon promotions to customers. These advertisements and promotions will be based on the past purchases of existing customers. The advertisements and promotions will target buying patterns by one or more of these demographics: gender, age, postal code, and region. Most of the promotions will be based on gender and age. Queries will be used to retrieve the customer demographics information.

You want the query response time to be as fast as possible. What should you do?

A. Add indexes on the **PostalCode**, **State**, and **DateOfBirth** columns of the **Customers** table.
B. Denormalize the **Customers** table
C. Create a view on the **Customers**, **SalesLineItem**, **State**, and **Product** tables.
D. Create a function to return the required data from the **Customers** table.

Answer: B.
Explanation: If a database is over-normalized, i.e. the database is defined with numerous, small, interrelated tables, database performance is reduced. This is because when the database processes the data in numerous, small, interrelated tables, it has to combine the related data, which results in an increase in the database's workload. In these situations, denormalizing the database slightly to simplify complex processes can improve performance.

In this scenario, the denormalizing could be accomplished by adding an extra column to the Customers table and use that column to track past purchases.

Incorrect answers:
A: Adding indexes to columns can be beneficial, especially if the keys are used in join operations. These keys often, almost all, have foreign key constraints.

However, the PostalCode, State, and DateOfBirth columns are not likely to have foreign key constraints or to be used in joins. Creating indexes on these columns would not improve performance. Probably performance would decrease, since adding indexes increases the size of the database.

C: Create a view of the tables would not improve performance. The joins of the tables would still have to be made.

D: A function can be used to simplify the coding of some queries. It cannot, however, improve performance.

Q. 96
You are a database developer for Lucerne Publishing. You are designing a human resources database that contains tables named Employee and Salary.

You interview users and discover the following information:

- **The Employee table will often be joined with the Salary table on the EmployeeID column.**
- **Individual records in the Employee table will be selected by social security number (SSN).**
- **A list of employees will be created. The list will be produced in alphabetical order by last name, and then followed by first name.**

You need to design the indexes for the tables while optimizing the performance of the indexes.
Which three scripts should you use? (Each correct answer presents part of the solution. Choose three.)

A. CREATE CLUSTERED INDEX [IX_EmployeeName] ON [dbo].[Employee] ([LastName], [FirstName])

B. CREATE INDEX [IX_EmployeeFirstName] ON [dbo].[Employee] ([First Name])
CREATE INDEX [IX_EmployeeLastName] ON [dbo].[Employee] ([Last Name])

C. CREATE UNIQUE INDEX [IX_EmployeeEmployeeID] ON [dbo].[Employee] ([EmployeeID])

D. CREATE UNIQUE INDEX [IX_EmployeeSSN] ON [dbo].[Employee] ([SSN])

E. CREATE CLUSTERED INDEX [IX_EmployeeEmployeeID] ON [dbo].[Employee] ([EmployeeID])

F. CREATE CLUSTERED INDEX [IX_EmployeeSSN] ON [dbo].[Employee] ([SSN])

Answer: A, C, D.
Explanation: We need to create three indexes meeting the requirements of this scenario. The selection of the clustered index will be the most critical decision. Some facts on indexes:

Á Only one index in a table can be a clustered index.
Á A clustered index determines the physical ordering of the data for that table. The table will be physically ordered by the indexed column(s).
Á The clustered index is included in all non-clustered indexes.
Á Nonclustered indexes are the optimal choice for exact match queries because the index contains entries describing the exact location in the table of the data values being searched for in the queries.

Compiling list
The reports of employees will be ordered in alphabetical order by last name, and then followed by first name. By choosing a clustered index on the LastName and FirstName columns, the table would be sorted in the correct order. (Answer A.)

Joining of tables
The Employee table is often joined with the Salary table on the EmployeeID column. We use a unique unclustered key on the EmployeeID column. (Answer C.)

Retrieving individual records
Individual records will be selected by the SSN column. An unclustered index should be created on the SSN column. (Answer D.)

Incorrect answers:
B: Instead of creating two separate unclustered indexes, we create a composite clustered index out of both these columns. It would benefit the reports.

C: It's better to create a composite clustered index on both the LastName and FirstName columns. Creating a clustered index on the EmployeeID would benefit the joins somewhat. However, it would be better to put the clustered index on the LastName and the FirstName columns.

F: The SSN column is used for retrieval, so there is no need of a clustered index.

Q. 97
You are a database developer for a large electric utility company. The company is divided into many departments, and each employee of the company is assigned to a department. You create a table named Employee that contains information about all employees, including the department to which they belong. The script that was used to create the Employee table is shown in the exhibit.

```
CREATE TABLE Employee
            (
            EmployeeID uniqueidentifier NOT NULL,
            FirstName char (20) NOT NULL,
            LastName char (25) NOT NULL,
            DepartmentID int NOT NULL,
            Salary money NOT NULL,
            CONSTRAINT PK_Employee PRIMARY KEY (EmployeeID)
            )
```

Each department manager should be able to view only the information in the Employee table that pertains to his or her department. What should you do?

A. Use GRANT, REVOKE, and DENY statements to assign permissions to each department manager.

B. Add the database login of each department manager to the **db_datareader** fixed database role.

C. Build tables and views that enforce row-level security on the **Employee** table.

D. Use SQL Server Enterprise Manager to assign permissions on the **Employee** table.

Answer: C.
Explanation: Views can be used with WHERE clauses to limit the rows. A view for each department could be built, and the department manager would only be granted access to the view that shows information for his department. This would be provided by using the WITH CHECK option.

Incorrect answers:
A: We must use views, not tables, to assign different permissions to different subsets of the table.

B: The **db_datareader** role would allow the managers to read every table in the database, but we want to restrict them.

D: We must use views, not tables, to assign different permissions to different subsets of table.

Q. 98
You are a database developer for your company's Human Resources database. This database includes a table named Employee that contains confidential ID numbers and salaries. The table also includes non-confidential information, such as employee names and addresses.

You need to make all the non-confidential information in the Employee table available in XML format to an external application. The external application should be able to specify the exact format of the XML data. You also need to hide the existence of the confidential information from the external application.

What should you do?

A. Create a stored procedure that returns the non-confidential information from the **Employee** table formatted as XML.

B. Create a user-defined function that returns the non-confidential information from the **Employee** table in a rowset that is formatted as XML.

C. Create a view that includes only the non-confidential information from the **Employee** table.
Give the external application permission to submit queries against the view.

D. Set column-level permissions on the **Employee** table to prevent the external application from viewing the confidential columns.
Give the external application permissions to submit queries against the table.

Answer: C.
Explanation:
The requirement was that the information had to be available in XML format to the external application. By giving the external application permission to query a view, it will be presented with data in a relational form, not in an XML form. But you can create XML views of relational data using XDR (XML-Data Reduced) schemas. These views can then be queried using XPath queries. The external application would be able to specify th format of the XML data. This is similar to creating views using CREATE VIEW statements and specifying SQL queries against the view.

Note: Regarding the FOR XML statement (which is not good enough in this scenario): You can execute SQL queries against existing relational databases in order to return a result as an XML document rather than as a standard rowset. To retrieve a result directly, use the FOR XML clause of the SELECT statement.

The FOR XML clause uses the following syntax:

FOR XML {RAW | AUTO | EXPLICIT} [, XMLDATA] [, ELEMENTS][, BINARY BASE64]

The problem here is how to allow the external application to specify the exact format of the XML data. This seems not to be possible. Neither stored procedures nor user-defined functions are able to provide the required result.

Reference: BOL, Basic Syntax of the FOR XML Clause

Incorrect answers:

A: Although the stored procedure could be used to return the only non-confidential column, we cannot specify the exact XML format, as the external application should be allowed to do this.

B: Although the stored procedure could be used to return the only non-confidential column, we cannot specify the exact XML format, as the external application should be allowed to do this.

D: Although it is possible to set column-level permissions on the Employee table to prevent the external application from viewing the confidential columns, the creation of a view that performs the same function is preferred.

Q. 99

You are a database developer for Tailspin Toys. You have two SQL Server 2000 computers named CORP1 and CORP2. Both of these computers use SQL Server Authentication. CORP2 stores data that has been archived from CORP1. At the end of each month, data is removed from CORP1 and transferred to CORP2.

You are designing quarterly reports that will include data from both CORP1 and CORP2. You want the distributed queries to execute as quickly as possible.

Which three actions should you take? (Each correct answer presents part of the solution. Choose Three.)

A. Create a stored procedure that will use the OPENROWSET statement to retrieve the data.

B. Create a stored procedure that will use the fully qualified table name on CORP2 to retrieve the data.

C. Create a script that uses the OPENQUERY statement to retrieve the data.

D. On CORP1, execute the **sp_addlinkedserver** system stored procedure.

E. On CORP1, execute the **sp_addlinkedsrvlogin** system stored procedure.

F. On CORP2, execute the **sp_serveroption** system stored procedure and set the **collation compatible** to ON.

Answer: B, D, E.
Explanation:
B. Objects in these linked servers can be referenced in Transact-SQL statements using four-part names. The linked server name can also be specified in an OPENQUERY statement to open a rowset from the OLE DB data source. This rowset can then be referenced like a table in Transact-SQL statements; for example, in a stored procedure.

D: You can set up a linked server with the sp_addlinkedserver system stored procedure.

E: The sp_addlinkedsrvlogin stored procedure creates or updates a mapping between logins on the local instance of SQL Server 2000 and remote logins on the linked server.

The sp_addlinkedserver and sp_addlinkedsrvlogin system stored procedures are used to give a server name to an OLE DB data source.

Incorrect answers:
A: The OPENROWSET or OPENDATASOURCE functions can be specified together with the information needed to connect to the linked server when infrequent references to a data source is required. The rowset can then be referenced the same way a table is referenced in Transact-SQL statements.

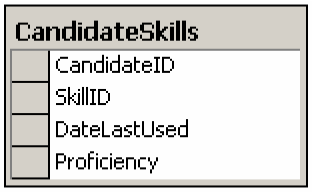
You should use the OPENROWSET and OPENDATASOURCE functions only to reference OLE DB data sources that are accessed infrequently. For any data sources that will be accessed more than a few times, define a linked server. Neither OPENDATASOURCE nor OPENROWSET provide all of the functionality of linked server definitions, such as security management or catalog information queries. Each time these functions are called, all connection information, including passwords, must be provided.

C: OPENQUERY statements are used to create ad hoc queries. It would be more productive to use a stored procedure that uses fully qualified names to retrieve data on the linked server.

F: The sp_serveroption stored procedure is not crucial for setting up a linked server.

Note: The sp_serveroption stored procedure sets server options for remote servers and linked servers. The **collation compatible** option affects Distributed Query execution against linked servers. If this option is set to **true**, SQL Server 2000 assumes that all characters in the linked server are compatible with the local server, with regard to character set and collation sequence. This enables SQL Server to send comparisons on character columns to the provider. If this option is not set, SQL Server always evaluates comparisons on character columns locally. This option should be set only if it is certain that the data source corresponding to the linked server has the same character set and sort order as the local server. Since CORP2 holds archived data from CORP1, the data will have the same character set and sort order on both servers

Q. 100
You are a database developer for an IT consulting company. You are designing a database to record information about potential consultants. You create a table named CandidateSkills for the database. The table is shown in the exhibit.



How should you uniquely identify the skills for each consultant?

A. Create a PRIMARY KEY constraint on the **CandidateID** column.

B. Create a PRIMARY KEY constraint on the **CandidateID** and **DateLastUsed** columns.

C. Create a PRIMARY KEY constraint on the **CandidateID** and **SkillID** columns.

D. Create a PRIMARY KEY constraint on the **CandidateID**, **SkillID**, and **DateLastUsed** columns.

Answer: C.
Explanation: A PRIMARY KEY constraint enforces the uniqueness of data entered in specified columns that do not allow null values. Duplicate data values are allowed in a specific column if a PRIMARY KEY consists of more than one column, but each combination of data values from all the columns in the PRIMARY KEY must be unique. In this scenario we are required to create a PRIMARY KEY constraint that will uniquely identify the skills of potential consultants. We place the data of each potential consultant in the CandidateID column and the corresponding data pertaining to their skills in the SkillID column. Thus we would require a PRIMARY KEY constraint on the CandidateID and SkillID columns.

Incorrect answers:

A: A PRIMARY KEY constraint placed on the CandidateID would only enforce the uniqueness of the potential consultant. It will not uniquely relate the potential candidate data to the potential candidate's skill.

B: Creating a PRIMARY KEY constraint on the CandidateID and DateLastUsed columns would enforce the uniqueness of the combination of data in the CandidateID and DateLastUsed columns. It will not enforce the uniqueness of the combination of data in the CandidateID and SkillID columns and would thus not meet the requirements set in this scenario.

D: Creating a PRIMARY KEY constraint on the CandidateID, SkillID, and DateLastUsed columns would enforce the uniqueness of the combination of data in the CandidateID, SkillID and DateLastUsed columns. However, the scenario does not require us to uniquely identify the data in the DateLastUsed column. Therefore Answer C would be the best answer.

Q. 101
You are a database developer for Proseware, Inc. You are implementing a database for the database of the company's human resources department. This database will store employee information. You create a table named EmployeeContact that contains the following columns:

HomePhone, BusinessPhone, FaxNumber, and EmailAddress

You must ensure that each record contains a value for either the HomePhone column or the BusinessPhone column. What should you do?

A. Create a rule that disallows null values.
Bind the rule to both the **HomePhone** and **BusinessPhone** columns.

B. Create a rule that prevents null values from being entered into both the **HomePhone** and **BusinessPhone** columns.
Bind the rule to the table.

C. Add CHECK constraints on the **HomePhone** and **BusinessPhone** columns that prevent null values from being entered into the columns.

D. Add a CHECK constraint on the table to validate that at least one of the values entered into the **HomePhone** and **BusinessPhone** columns is non-null.

E. Create a trigger that counts the number of items entered without a value in the **HomePhone** column and then counts the number of items entered without a value in the **BusinessPhone** column.
Configure the trigger so that if one or more rows are found that meet these conditions, the trigger will cancel the data modification.

Answer: D.
Explanation: We want to check rows so that rows cannot have NULL values in both the HomePhone columns. One NULL value in either column is OK, though.

We need to create one constraint mechanism that takes both columns into account. We should add a CHECK table constraint that prevents NULL values from being entered into both mentioned columns. This could be done by:

CONSTRAINT CK_phone CHECK (NOT (HomePhone is NULL and BusinessPhone is NULL))

Note that rules can be bound to columns but not to tables. So we cannot use rules in this scenario.

A general note on rules and check constraints:
Rules perform the same function as CHECK constraints and are used in cases where backward compatibility is required. CHECK constraints are the preferred, standard way of restricting the values in a column and are also more concise than rules. Multiple CHECK constraints can be applied to a column, while only one rule can be applied per column. CHECK constraints are also specified as part of the CREATE TABLE statement while rules are created as separate objects and then bound to the column.

Incorrect answers:

A: This solution is too simplistic. This rule requires that both columns are non-NULL, but the requirement is that only one of them is non-NULL.

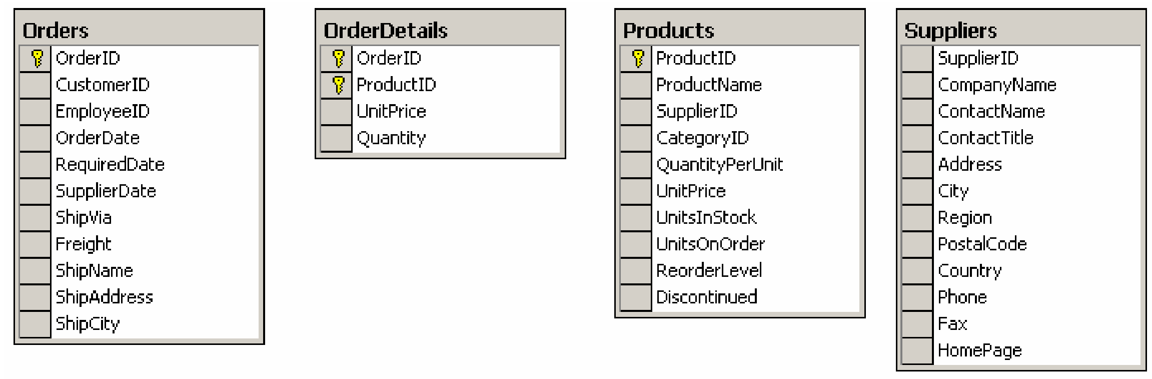
B: We need a restriction on the table, not on particular columns. Rules can be bound to columns but not to tables, so they cannot be used here.

C: This CHECK constraint would enforce non-NULL values in both columns, but we want to allow a NULL value in one of the columns as long as the other column is not NULL.

E: A trigger could solve this problem. However, triggers require more system resources and should only be used if no simple solution exists. A constraint is the preferred solution.

Note: Constraints and triggers have different benefits making them useful in different situations. Triggers can contain complex processing logic that uses Transact-SQL code and can therefore support all the functionality of constraints. However, triggers are not always the best method for a given situation. It is recommended that entity integrity be enforced at the lowest level by indexes that are part of PRIMARY KEY and UNIQUE constraints, that domain integrity be enforced through CHECK constraints, and that referential integrity be enforced through FOREIGN KEY constraints where these features meet the functional needs of the application. Triggers should only be used when the features supported by the other constraint types cannot meet the functional needs of the application. In this scenario a CHECK constraint applied at the table level can meet the functional needs of the application and would be the preferred method.

Q. 102
You are designing an inventory and shipping database for Contoso, Ltd. You create the logical database design shown in the exhibit.



You must ensure that the referential integrity of the database is maintained. Which three types of constraints should you apply to your design? (Each correct answer presents part of the solution. Choose all that apply.)

A. Create a FOREIGN KEY constraint on the **Products** table that references the **OrderDetails** table.

B. Create a FOREIGN KEY constraint on the **Products** table that references the **Suppliers** table.

C. Create a FOREIGN KEY constraint on the **Orders** table that references the **OrderDetails** table.

D. Create a FOREIGN KEY constraint on the **OrderDetails** table that references the **Order** table.

E. Create a FOREIGN KEY constraint on the **OrderDetails** table that references the **Products** table.

F. Create a FOREIGN KEY constraint on the **Suppliers** table that references the **Products** table.

Answer: B, D, E.
Explanation:

From a logical database design line of thought:
The tables are entities and they should be connected by relations. In particular, one-to-many should be used in almost every case. Occasionally, one-to-one relations are used, but it is very rare.

The Suppliers table
Every supplier can supply one or more products.
This is a one-to-many relation from the Suppliers table to the Products table.
The one-to-many relationship is implemented by creating a foreign key constraint on the SupplierID column in the Products table referencing the primary key column SupplierID in the Suppliers table. (Answer B.)

The Products table
Every product could be included in one or several orders.
Every order could include one or several products.
So we have a many-to-many relationship between the Orders and Products table. Since SQL Server, and almost every other database management system, doesn't support many-to-many relations, the extra table OrderDetails has been added. The OrderDetails table will be used to relate the Orders and Products table with each other. We need two one-to-many relations:

∉ One one-to-many relation from the Products table to the OrderDetails table. This relationship will be implemented by a foreign key constraint on the ProductID column of the OrderDetails table referencing the ProductID primary key column of the Product table. (Answer D.)

∉ One one-to-many relation from the Orders table to the OrderDetails table. This relationship will be implemented by a foreign key constraint on the OrderID column of the OrderDetails referencing the OrderID primary key column of the Orders table. (Answer E.)

From a procedural line of thought:
The OrderDetails table would have a FOREIGN KEY against the Orders table and the Products table so that the Orders table and the Products table would be updated, once an order for a particular product or products have been placed. The Products table would have a FOREIGN KEY against a Suppliers table to update the Suppliers table once products are received.

Incorrect answers:
A: **Logical design argument**
A FOREIGN KEY constraint on the Products table that references the OrderDetails table is a one-to-many relation from the OrderDetails to the Products table, which is to say that one OrderDetail row can be included in many rows in the Products table, but we want the opposite. We want a one-to-many relation from the Products table to the OrderDetails table.

Procedural argument
Creating a FOREIGN KEY constraints on the Products table that references the Order Details table would result in the Order Details table being updated when products are received and entered into the Products table.

C: **Logical design argument**
We want a one-to-many relation from the Orders to the OrderDetails table, not the other way around. We want that every order row should have one or several corresponding rows in the OrderDetails table, not the other way around.

Procedural argument
Incoming orders would be entered into the OrderDetails table and not the Orders table, thus the OrderDetails table has a FOREIGN KEY against the Orders table so that the Orders table would be updated once data is entered into the OrderDetails table.

F: **Logical design argument**
Every supplier can supply one or several products, so we need a one-to-many relation from the Suppliers table to the Products table. This relationship is implemented with a foreign key constraint on a column in the Products table referencing a column in the Suppliers table, not the other way around.

Procedural argument
Data pertaining to products received would be entered into the Products table, after which the Suppliers table would be updated, thus the Products table would have a FOREIGN KEY against a Suppliers table to update the Suppliers table once products are received.

Q. 103
You are a database developer for a SQL Server 2000 database. You have created a stored procedure to produce the EndOfMonthSales report for the sales department.

You issue the following statement to assign permissions to the EndOfMonthSales report:

```
GRANT EXECUTE ON EndOfMonthSales TO SalesDept
```

Andrea has just joined the sales department and is a member of the SalesDept role. Andrea is also a member of the Marketing role, which has been denied access to the EndOfMonthSales report.

Andrea is unable to execute the stored procedure. No other sales department employees are experiencing this problem.

What should you do?

A. Add Andrea to the **db_datareader** database role.
Grant REFERENCES permissions to the public role.

B. Remove Andrea from the **Marketing** database role.

C. Grant Andrea's database user account SELECT permissions on the tables referenced in the stored procedure.

D. Grant Andrea's database user account REFERENCES permissions on the tables referenced in the stored procedure.

Answer: B.
Explanation: When users have multiple permissions to a resource on the basis of their membership in multiple roles, and where the permissions conflict, the most restrictive permission is used. In this scenario Andrea is a member of the SalesDept role, which has execute permission to the stored procedure. She is also a member of the Marketing role which has a deny permission to the stored procedure. As the deny permission is the most restrictive, it overrides the execute permission Andrea has as a member of the SalesDept role. Removing Andrea from the Marketing database role will also remove the deny permission and Andrea will be able to use the stored procedure.

Note: A user only needs execute permission to use a stored procedure. There are no other permissions that can be granted to stored procedures. The users don't need permissions to the underlying tables and views referenced by the stored procedure.

Incorrect answers:

A: We must modify Andrea's permission on the stored procedure. This cannot be accomplished with database roles and REFERENCES permissions.

Note: The db_datareader fixed database role allows its members to SELECT all data from any user table in the database, while the REFERENCES permission allows the owner of one table to use columns in another table as the target of a REFERENCES FOREIGN KEY constraint if they have REFERENCES permissions on that table. These permissions are table object permissions.

C: Granting Andrea's database user account SELECT permissions on the tables referenced in the stored procedure will not enable her to use the stored procedure, as she is still denied EXECUTE permission to the stored procedure by virtue of her membership in the Marketing role. Furthermore, grant SELECT permissions are restricted to table level objects.

D: Granting Andrea's database user account REFERENCES permissions on the tables referenced in the stored procedure will not enable her to use the stored procedure, as she is still denied EXECUTE permission to the stored procedure by virtue of her membership to the Marketing role. Furthermore, the REFERENCES permission allows the owner of one table to use columns in another table as the target of a REFERENCES FOREIGN KEY constraint if they have REFERENCES permissions on that table. These permissions are table object permissions.

Q. 104
You are a database developer for a large travel company. You have been granted CREATE VIEW permissions in the Reservations database. Your co-worker, Eric, has been granted CREATE TABLE permissions. Neither of you have been given database owner or system permissions, nor have you been added to any fixed server roles.

Eric has created a table named Traveler that holds information about your company's customers. This table is shown in the exhibit. .

Travel agents will connect to the database and view the information stored in this table. The database logins for the travel agents have been assigned to the Agent database role.

You want the travel agents to be able to view the name and address information from the Traveler table in two columns instead of six. One column should contain the traveler name and the other column should contain the address.

Which three actions should you take? (Each correct answer presents part of the solution. Choose three)

A. Grant the Agent role SELECT permissions on the Traveler table.

B. Instruct Eric to grant the Agent role SELECT permissions on the Traveler table.

C. Instruct Eric to grant you REFERENCES permissions on needed columns in the Traveler table.

D. Instruct Eric to create a view named vwTravelers that displays the data in the desired format.

E. Create a view named vwTravelers that displays the data in the desired format.

F. Grant the Agent role SELECT permissions on the vwTravelers view.

Answer: C, E, F.
Explanation: Eric can create tables and you can create views. We must ensure that the Agent role can use a view. To avoid the broken ownership chain, we must explictely give permission to the Agent role for both the view and the table.

C: By granting us REFERENCES permission on the selected columns of the table we would be able to create a view on the table.

E: You must create the view. Eric does not have the right to create views.

F: As the owner of the view, you give the Agent role to the view.

Incorrect answers:

A: The creator of a view cannot grant permissions to database objects other than for the view that they own. You cannot grant the Agent role SELECT permissions on the Traveler table.

B: The Agent role must only have access to two columns in the table. If we give the agent role SELECT permission to the whole table, they would be able to access all columns.

D: Eric has permission to create tables, but (it seems like) he does not have any permission to create views.

Q. 105

You are a database developer for a food wholesaler. Each week, the company fulfills orders from various customers. Normally, each customer orders the same quantity of certain items each week. Occasionally, the quantity of an item that a customer orders is significantly less than the customer's usual quantity. The information about each order is stored in a table named Invoice, which is located in a SQL Server 2000 database. The script that was used to create this table is shown in the exhibit.

EXHIBIT

```
CREATE TABLE Invoice
              (
              InvoiceID int NOT NULL,
              InvoiceNumber char(10) NOT NULL,
              CustomerName char(30) NOT NULL,
              InvoiceAmount money NOT NULL DEFAULT (0),
              CONSTRAINT PK_Invoice PRIMARY KEY (InvoiceID)
              )
```

You want to identify any pattern to these unusual orders. To do this, you need to produce a list of the invoices for each customer that are for a lesser amount than average invoice amount for that customer.

Which query should you use?

A.
```
SELECT i1.InvoiceNumber, i1.CustomerName, i1.InvoiceAmount
FROM Invoice As i1, Invoice AS i2
GROUP BY i1.InvoiceNumber, i1.CustomerName, i1.InvoiceAmount
HAVING i1.InvoiceAmount < AVG (i2.InvoiceAmount)
ORDER BY i1.CustomerName, i1.InvoiceNumber
```

B.
```
SELECT i1.InvoiceNumber, i1.CustomerName, i1.InvoiceAmount
FROM Invoice As i1
WHERE i1.InvoiceAmount <
```

```
                    (SELECT AVG (i2.InvoiceAmount)
                    FROM Invoice AS i2
                    WHERE i2.CustomerName=i1.CustomerName)
          ORDER BY i1.CustomerName, i1.InvoiceNumber
```

C.
```
SELECT i1.InvoiceNumber, i1.CustomerName, i1.InvoiceAmount
FROM Invoice As i1
WHERE i1.InvoiceAmount <
          (SELECT AVG (i2.InvoiceAmount)
          FROM Invoice AS i2)
ORDER BY i1.CustomerName, i1.InvoiceNumber
```

D.
```
SELECT i1.InvoiceNumber, i1.CustomerName, i1.InvoiceAmount,
     CASE WHEN i1.InvoiceAmount < AVG (i2.InvoiceAmount)
     THEN i1.InvoiceAmount ELSE 0 END
FROM Invoice As i1 INNER JOIN Invoice AS i2
ON i1.CustomerName = i2.CustomerName
GROUP BY i1.InvoiceNumber, i1.CustomerName, i1.InvoiceAmount
ORDER BY i1.CustomerName, i1.InvoiceNumber
```

Answer: B.
Explanation: The subquery is correct:
```
SELECT AVG (i2.InvoiceAmount)
FROM Invoice AS i2
WHERE i2.CustomerName=i1.CustomerName)
```
The first WHERE clause makes the correlated subquery calculate the average for the current CustomerName, not the average for the whole of the table. This is used since we want to catch the rows where the customer orders are significantly less than the customer's usual quantity.

Incorrect answers:

A: This code makes the result set include all rows where the InvoiceAmount is less than the average InvoiceAmount. But we are only interested in the orders where the InvoiceAmount is less than the average for this particular customer. Instead of using a HAVING clause we must use a correlated subquery.

C: This solution doesn't give the correct result. We are only interested in the orders where the InvoiceAmount is less than the average for this particular customer. This query gives the orders that are less than the average of all orders.

D: Here a subquery of sorts is defined in the SELECT list using a CASE WHEN construct. This is possible, but the query is unnecessarily complicated. A simple subquery would be more elegant and efficient.

Q. 106
You are a database developer for Contoso, Ltd. The company stores its sales data in a SQL Server 2000 database. The database contains a table named Customers, which has 500,000 rows. The script that was used to create the Customers table is shown in the exhibit.

```
CREATE TABLE Customers
(
CustomerID int IDENTITY NOT NULL,
CompanyName varchar(40) NOT NULL,
ContactName varchar(30) NULL,
ContactTitle varchar(30) NULL,
Address varchar(60) NULL,
City varchar(15) NULL,
Region varchar(15) NULL,
PostalCode varchar(10) NULL,
Country varchar(15) NULL,
Phone varchar(24) NULL,
Fax varchar(24) NULL,
CONSTRAINT PK_Customers PRIMARY KEY CLUSTERED (CustomerID)
)
```

Many critical queries are executed against the table, which select customer records based on the City and Region columns. Very few customers live in the same city as each other, but many live in same region as each other.

How should you optimize the performance of these queries?

A. Create a view on the **Customers** table.

B. Create a function that returns data for the **Customers** table.

C. Create a composite, nonclustered index on the **City** and **Region** columns, and use **City** as the first column in the index.

D. Create a composite, nonclustered index on the **City** and **Region** columns, and use **Region** as the first column in the index.

E. Add the **City** and **Region** columns to the clustered index, and use **Region** as the first column in the index.

Answer: C.
Explanation:
Many critical queries are run against the table, which select customer records based on the City and Region columns. To improve performance we should index these two columns. We should choose a composite index that only includes these two columns. Then we must decide the ordering of the columns in the index. We should choose the first column to be the column that has most distinct different values. This allows SQL Server to find the right row faster. In our scenario it is stated that very few customers live in the same city as each other. The city column contains the most distinct values and should be chosen as the first column in the index.

Incorrect answers:
A: A view is used to restrict a user to specific rows in a table, to restrict a user to specific columns, to join columns from multiple tables so that they look like a single table, or to aggregate information instead of supplying details. Views do not aid in improving query performance.

B: Although a function can return a rowset, it will not improve query time.

D: For best performance a composite index should be constructed by selecting the column with the most distinct values as the leftmost column. The City column, not the Region column, should be first column in the index. The scenario states that very few customers live in the same city, but many live in the same region.

E: Although a clustered index is preferred to a nonclustered index when the queries must return a range of values, as is the case in this scenario, it is important to define the clustered index key with as few columns as possible. If a large clustered index key is defined, any nonclustered indexes that are defined on the same table will be significantly larger because the nonclustered index entries contain the clustering key.

Q. 107
You are a database developer for a SQL Server 2000 database. The database is in the default configuration. The number of users accessing the database has increased from 100 to 1,000 in the last month. Users inform you that they are receiving error messages frequently. The following is an example of an error message that was received:

```
Transaction (Process ID 56) was deadlocked on [lock] resources with another
process and has been chosen as the deadlock victim. Rerun the transaction.
```

What should you do?

A. Use a higher transaction isolation level for transactions used in the database.

B. Use SQL Profiler to capture deadlock events.

C. Use System Monitor to monitor lock requests and deadlocks.

D. Execute the **sp_configure** system stored procedure to increase the number of simultaneous user connections allowed to SQL server.

Answer: B.
Explanation:
First we should try to prevent deadlocks, for instance by:

- making all processes access resources in a consistent order
- reducing the transaction isolation level if it's suitable for the application

If the reason of the deadlock isn’t apparent, there are several tools that can be used to track the source of the deadlock:

- The graphical lock display in SQL Server Enterprise Manager. Not listed here, though it would probably the first place to look to get some initial information of the deadlocks.
- System Monitor. Here you get statistical information on the locks. It is good for monitoring locks but not a good tool to find the reasons behind the deadlocks.
- Examine Trace Flag 1204. Not a listed option here.
- The SQL Profiler, which is the best solution provided here. SQL Profiler provides information for basic deadlock detection.

Note: 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 the lock monitor thread. After a deadlock is identified, SQL Server ends a 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.

Incorrect answers:
A: One method of preventing cycle deadlocks is by lowering, not increasing, the transaction isolation level if it's suitable for the application.

C: SQL Server 2000 provides objects and counters that can be used by System Monitor (or Performance Monitor in Windows NT 4.0) to monitor activity in computers running an instance of SQL Server. Each object contains one or more counters that determine various aspects of the objects to monitor. The Locks object provides information about SQL Server locks and deadlocks on individual resource types.

However, the System Monitor only provides statistical information and the possibility of raising alerts when predefined thresholds are reached.

D: Increasing the number of concurrent user connections to SQL Server increases the possibility of the occurrence of deadlock events. It would be better to analyze the cause of the deadlocks. SQL Profiler can be used for this purpose.