CERT24.COM

100% Success Guarantee of IT Exams

-The fastest and guaranteed way to certy now!

http://cert24.com/

Exam Number/Code:70-761

Exam Name: Querying Data with Transact-SQL

Version: Demo

You create a table named Products by running the following Transact-SQL statement:

```
CREATE TABLE Products (
      ProductID int IDENTITY(1,1) NOT NULL PRIMARY KEY,
      ProductName nvarchar(100) NULL,
      UnitPrice decimal(18, 2) NOT NULL,
      UnitsInStock int NOT NULL,
      UnitsOnOrder int NULL
)
```

You have the following stored procedure:

```
CREATE PROCEDURE InsertProduct
     @ProductName nvarchar(100),
     @UnitPrice decimal(18,2),
     @UnitsInStock int,
     @UnitsOnOrder int
AS
BEGIN
     INSERT INTO Products(ProductName,ProductPrice,ProductsInStock,ProductsOnOrder)
     VALUES (@ProductName,@UnitPrice,@UnitsInStock,@UnitsOnOrder)
END
```

You need to modify the stored procedure to meet the following new requirements:
Insert product records as a single unit of work.
Return error number 51000 when a product fails to insert into the database. If a product record insert operation fails, the product information must not be permanently written to the database.

Solution: You run the following Transact-SQL statement:

```
ALTER PROCEDURE InsertProduct
@ProductName nvarchar(100),
@UnitPrice decimal(18,2),
@UnitsInStock int,
@UnitsOnOrder int
AS
BEGIN
   SET XACT_ABORT ON
   BEGIN TRY
      BEGIN TRANSACTION
         INSERT INTO Products(ProductName,ProductPrice,ProductsInStock,ProductsOnOrder)
         VALUES (@ProductName,@UnitPrice,@UnitsInStock,@UnitsOnOrder)
      COMMIT TRANSACTION
   END TRY
   BEGIN CATCH
      IF XACT_STATE() <> 0 ROLLBACK TRANSACTION
      THROW 51000, 'The product could not be created.', 1
   END CATCH
END
```

Does the solution meet the goal?

A. Yes
B. No

Answer: B

Explanation:
With X_ABORT ON the INSERT INTO statement and the transaction will be rolled back when an error is raised, it would then not be possible to ROLLBACK it again in the IF XACT_STATE() <> O ROLLBACK TRANSACTION statement.
Note: A transaction is correctly defined for the INSERT INTO ..VALUES statement, and if there is an error in the transaction it will be caughtant he transaction will be rolled back, finally an error 51000 will be raised.
Note: When SET XACT_ABORT is ON, if a Transact-SQL statement raises a run-time error, the entire transaction is terminated and rolled back.
XACT_STATE is a scalar function thatreports the user transaction state of a current running request.
XACT_STATE indicates whether the request has an active user transaction, and whether the transaction is capable of being committed.
The states of XACT_STATE are:
0 There is no active user transaction for the current request.
1 The current request has an active user transaction. The request can perform any actions, including writing data and committing the transaction.
2 The current request has an active user transaction, but an error hasoccurred that has caused the transaction to be classified as an uncommittable transaction.
References:
https://msdn.microsoft.com/en-us/library/ms188792.aspx
https://msdn.microsoft.com/en-us/library/ms189797.aspx

Question 2
You create a table named Products by running the following Transact-SQL statement:

```
CREATE TABLE Products (
      ProductID int IDENTITY(1,1) NOT NULL PRIMARY KEY,
      ProductName nvarchar(100) NULL,
      UnitPrice decimal(18, 2) NOT NULL,
      UnitsInStock int NOT NULL,
      UnitsOnOrder int NULL
)
```

You have the following stored procedure:

```
CREATE PROCEDURE InsertProduct
     @ProductName nvarchar(100),
     @UnitPrice decimal(18,2),
     @UnitsInStock int,
     @UnitsOnOrder int
AS
BEGIN
     INSERT INTO Products(ProductName,ProductPrice,ProductsInStock,ProductsOnOrder)
     VALUES (@ProductName,@UnitPrice,@UnitsInStock,@UnitsOnOrder)
END
```

You need to modify the stored procedure to meet the following new requirements:
Insert product records as a single unit of work.
Return error number 51000 when a product fails to insert into the database. If a product record insert operation fails, the product information must not be permanently written to the database.

Solution: You run the following Transact-SQL statement:

```
ALTER PROCEDURE InsertProduct
@ProductName nvarchar(100),
@UnitPrice decimal(18,2),
@UnitsInStock int,
@UnitsOnOrder int
AS
BEGIN
   BEGIN TRY
      BEGIN TRANSACTION
         INSERT INTO Products(ProductName,ProductPrice,ProductsInStock,ProductsOnOrder)
         VALUES (@ProductName,@UnitPrice,@UnitsInStock,@UnitsOnOrder)
      COMMIT TRANSACTION
   END TRY
   BEGIN CATCH
      IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION
         IF @@ERROR = 51000
            THROW
      END CATCH
END
```

Does the solution meet the goal?

A. Yes
B. No

Answer: B

Explanation:
A transaction is correctly defined for the INSERT INTO ..VALUES statement, and if there is an error in the transaction it will be caught ant he transaction will be rolled back.
However, error number 51000 will not be returned, as it is only used in an IF @ERROR = 51000 statement.
Note: @@TRANCOUNT returns the number of BEGIN TRANSACTION statements that have occurred on the current connection.
References: https://msdn.microsoft.com/en-us/library/ms187967.aspx

Question 3
You create a table named Products by running the following Transact-SQL statement:

```
CREATE TABLE Products (
      ProductID int IDENTITY(1,1) NOT NULL PRIMARY KEY,
      ProductName nvarchar(100) NULL,
      UnitPrice decimal(18, 2) NOT NULL,
      UnitsInStock int NOT NULL,
      UnitsOnOrder int NULL
)
```

You have the following stored procedure:

```
CREATE PROCEDURE InsertProduct
     @ProductName nvarchar(100),
     @UnitPrice decimal(18,2),
     @UnitsInStock int,
     @UnitsOnOrder int
AS
BEGIN
     INSERT INTO Products(ProductName,ProductPrice,ProductsInStock,ProductsOnOrder)
     VALUES (@ProductName,@UnitPrice,@UnitsInStock,@UnitsOnOrder)
END
```

You need to modify the stored procedure to meet the following new requirements:
Insert product records as a single unit of work.
Return error number 51000 when a product fails to insert into the database. If a product record insert operation fails, the product information must not be permanently written to the database.

Solution: You run the following Transact-SQL statement:

```
ALTER PROCEDURE InsertProduct
@ProductName nvarchar(100),
@UnitPrice decimal(18,2),
@UnitsInStock int,
@UnitsOnOrder int
AS
BEGIN
     BEGIN TRY
          INSERT INTO Products(ProductName,ProductPrice,ProductsInStock,ProductsOnOrder)
               VALUES (@ProductName,@UnitPrice,@UnitsInStock,@UnitsOnOrder)
     END TRY
     BEGIN CATCH
          THROW 51000, 'The product could not be created.', 1
          END CATCH
END
```

Does the solution meet the goal?

A. Yes
B. No

Answer: A

Explanation:
If the INSERT INTO statement raises an error, the statement will be caught and an error 51000 will be thrown. In this case no records will have been inserted.
Note:
You can implement error handling for the INSERT statement by specifying the statement in a TRY...
CATCH construct.
If an INSERT statement violates a constraint or rule, or if it has a value incompatible with the data type of the column, the statement fails and an error message is returned.
References: https://msdn.microsoft.com/en-us/library/ms174335.aspx

Question 4
You create a table named Customer by running the following Transact-SQL statement:

```
CREATE TABLE Customer (
      CustomerID int IDENTITY(1,1) PRIMARY KEY,
      FirstName varchar(50) NULL,
      LastName varchar(50) NOT NULL,
      DateOfBirth date NOT NULL,
      CreditLimit money CHECK (CreditLimit < 10000),
      TownID int NULL REFERENCES dbo.Town(TownID),
      CreatedDate datetime DEFAULT(Getdate())
)
```

You must insert the following data into the Customer table:

Record	First name	Last name	Date of Birth	Credit limit	Town ID	Created date
Record 1	Yvonne	McKay	1984-05-25	9,000	no town details	current date and time
Record 2	Jossef	Goldberg	1995-06-03	5,500	no town details	current date and time

You need to ensure that both records are inserted or neither record is inserted.

Solution: You run the following Transact-SQL statement:

```
INSERT INTO Customer (FirstName, LastName, DateOfBirth, CreditLimit, CreatedDate)
VALUES ('Yvonne', 'McKay', '1984-05-25', 9000, GETDATE())
INSERT INTO Customer (FirstName, LastName, DateOfBirth, CreditLimit, CreatedDate)
VALUES ('Jossef', 'Goldberg', '1995-06-03', 5500, GETDATE())
GO
```

Does the solution meet the goal?

A. Yes
B. No

Answer: B

Explanation:
As there are two separate INSERT INTO statements we cannot ensure that both or neither records are inserted.

Question 5
You create a table named Customer by running the following Transact-SQL statement:

```
CREATE TABLE Customer (
      CustomerID int IDENTITY(1,1) PRIMARY KEY,
      FirstName varchar(50) NULL,
      LastName varchar(50) NOT NULL,
      DateOfBirth date NOT NULL,
      CreditLimit money CHECK (CreditLimit < 10000),
      TownID int NULL REFERENCES dbo.Town(TownID),
      CreatedDate datetime DEFAULT(Getdate())
)
```

You must insert the following data into the Customer table:

Record	First name	Last name	Date of Birth	Credit limit	Town ID	Created date
Record 1	Yvonne	McKay	1984-05-25	9,000	no town details	current date and time
Record 2	Jossef	Goldberg	1995-06-03	5,500	no town details	current date and time

You need to ensure that both records are inserted or neither record is inserted.

Solution: You run the following Transact-SQL statement:

```
INSERT INTO Customer (FirstName, LastName, DateOfBirth, CreditLimit, TownID, CreatedDate)
VALUES ('Yvonne', 'McKay', '1984-05-25', 9000, NULL, GETDATE())
INSERT INTO Customer (FirstName, LastName, DateOfBirth, CreditLimit, TownID, CreatedDate)
VALUES ('Jossef', 'Goldberg', '1995-06-03', 5500, NULL, GETDATE())
GO
```

Does the solution meet the goal?

A. Yes
B. No

Answer: B

Explanation:
As there are two separate INSERT INTO statements we cannot ensure that both or neither records are inserted.

Question 6
You create a table named Customer by running the following Transact-SQL statement:

```
CREATE TABLE Customer (
      CustomerID int IDENTITY(1,1) PRIMARY KEY,
      FirstName varchar(50) NULL,
      LastName varchar(50) NOT NULL,
      DateOfBirth date NOT NULL,
      CreditLimit money CHECK (CreditLimit < 10000),
      TownID int NULL REFERENCES dbo.Town(TownID),
      CreatedDate datetime DEFAULT(Getdate())
)
```

You must insert the following data into the Customer table:

Record	First name	Last name	Date of Birth	Credit limit	Town ID	Created date
Record 1	Yvonne	McKay	1984-05-25	9,000	no town details	current date and time
Record 2	Jossef	Goldberg	1995-06-03	5,500	no town details	current date and time

You need to ensure that both records are inserted or neither record is inserted.

Solution: You run the following Transact-SQL statement:

```
INSERT INTO dbo.Customer (FirstName, LastName, DateOfBirth, CreditLimit)
VALUES ('Yvonne', 'McKay', '1984-05-25', 9000), ("Jossef', 'Goldberg', '1995-06-03', 5500)
```

Does the solution meet the goal?

A. Yes
B. No

Answer: A

Explanation:
With the INSERT INTO..VALUES statement we can insert both values with just one statement. This ensures that both records or neither is inserted.
References:https://msdn.microsoft.com/en-us/library/ms174335.aspx

Question 7
You have a database that tracks orders and deliveries for customers in North America.
The database contains the following tables:

Sales.Customers

Column	Data type	Notes
CustomerID	int	primary key
CustomerCategoryID	int	foreign key to the Sales.CustomerCategories table
PostalCityID	int	foreign key to the Application.Cities table
DeliveryCityID	int	foreign key to the Application.Cities table
AccountOpenedDate	datetime	does not allow new values
StandardDiscountPercentage	int	does not allow new values
CreditLimit	decimal(18,2)	null values are permitted
IsOnCreditHold	bit	does not allow new values
DeliveryLocation	geography	does not allow new values
PhoneNumber	nvarchar(20)	does not allow new values data is formatted as follows: 425-555-0187

Application.Cities

Column	Data type	Notes
CityID	int	primary key
LatestRecordedPopulation	bigint	null values are permitted

Sales.CustomerCategories

Column	Data type	Notes
CustomerCategoryID	int	primary key
CustomerCategoryName	nvarchar(50)	does not allow null values

The company's development team is designing a customer directory application. The application must list customers by the area code of their phone number. The area code is defined as the first three characters of the phone number.
The main page of the application will be based on an indexed view that contains the area and phone number for all customers.
You need to return the area code from the PhoneNumber field.

Solution: You run the following Transact-SQL statement:

```
CREATE FUNCTION AreaCode (
     @phoneNumber nvarchar(20)
)
RETURNS
TABLE
WITH SCHEMABINDING
AS
RETURN (
     SELECT TOP 1 @phoneNumber as PhoneNumber, VALUE as AreaCode
     FROM STRING_SPLIT(@phoneNumber, '-')
)
```

Does the solution meet the goal?

A. Yes
B. No

Answer: B

Explanation:
The function should return nvarchar(10) and not a TABLE.
References: https://sqlstudies.com/2014/08/06/schemabinding-what-why/

Question 8
You have a database that tracks orders and deliveries for customers in North America.
The database contains the following tables:

Sales.Customers

Column	Data type	Notes
CustomerID	int	primary key
CustomerCategoryID	int	foreign key to the Sales.CustomerCategories table
PostalCityID	int	foreign key to the Application.Cities table
DeliveryCityID	int	foreign key to the Application.Cities table
AccountOpenedDate	datetime	does not allow new values
StandardDiscountPercentage	int	does not allow new values
CreditLimit	decimal(18,2)	null values are permitted
IsOnCreditHold	bit	does not allow new values
DeliveryLocation	geography	does not allow new values
PhoneNumber	nvarchar(20)	does not allow new values data is formatted as follows: 425-555-0187

Application.Cities

Column	Data type	Notes
CityID	int	primary key
LatestRecordedPopulation	bigint	null values are permitted

Sales.CustomerCategories

Column	Data type	Notes
CustomerCategoryID	int	primary key
CustomerCategoryName	nvarchar(50)	does not allow null values

The company's development team is designing a customer directory application. The application must list customers by the area code of their phone number. The area code is defined as the first three characters of the phone number.

The main page of the application will be based on an indexed view that contains the area and phone number for all customers.

You need to return the area code from the PhoneNumber field.

Solution: You run the following Transact-SQL statement:

```
CREATE FUNCTION AreaCode (
     @phoneNumber nvarchar(20)
)
RETURNS nvarchar(10)
AS
BEGIN
     DECLARE @areaCode nvarchar(max)
     SELECT TOP 1 @areaCode = VALUE FROM STRING_SPLIT(@phoneNumber, '-')
     RETURN @areaCode
END
```

Does the solution meet the goal?

A. Yes
B. No

Answer: B

Explanation:
As the result of the function will be used in an indexed view we should use schemabinding.
References: https://sqlstudies.com/2014/08/06/schemabinding-what-why/

Question 9
You have a database that tracks orders and deliveries for customers in North America.
The database contains the following tables:

Sales.Customers

Column	Data type	Notes
CustomerID	int	primary key
CustomerCategoryID	int	foreign key to the Sales.CustomerCategories table
PostalCityID	int	foreign key to the Application.Cities table
DeliveryCityID	int	foreign key to the Application.Cities table
AccountOpenedDate	datetime	does not allow new values
StandardDiscountPercentage	int	does not allow new values
CreditLimit	decimal(18,2)	null values are permitted
IsOnCreditHold	bit	does not allow new values
DeliveryLocation	geography	does not allow new values
PhoneNumber	nvarchar(20)	does not allow new values data is formatted as follows: 425-555-0187

Application.Cities

Column	Data type	Notes
CityID	int	primary key
LatestRecordedPopulation	bigint	null values are permitted

Sales.CustomerCategories

Column	Data type	Notes
CustomerCategoryID	int	primary key
CustomerCategoryName	nvarchar(50)	does not allow null values

The company's development team is designing a customer directory application. The application must list customers by the area code of their phone number. The area code is defined as the first three characters of the phone number.

The main page of the application will be based on an indexed view that contains the area and phone number for all customers.

You need to return the area code from the PhoneNumber field.

Solution: You run the following Transact-SQL statement:

```
CREATE FUNCTION AreaCode (
     @phoneNumber nvarchar(20)
)
RETURNS nvarchar(10)
WITH SCHEMABINDING
AS
BEGIN
     DECLARE @areaCode nvarchar(max)
     SELECT @areaCode = value FROM STRING_SPLIT(@phoneNumber, '-')
     RETURN @areaCode
END
```

Does the solution meet the goal?

A. Yes
B. No

Answer: B

Explanation:
We need SELECT TOP 1 @areacode =.. to ensure that only one value is returned.

Question 10
You query a database that includes two tables: Project and Task. The Project table includes the following columns:

Column name	Data type	Notes
ProjectId	int	This is a unique identifier for a project.
ProjectName	varchar(100)	
StartTime	datetime2(7)	
EndTime	datetime2(7)	A null value indicates the project is not finished yet.
UserId	int	Identifies the owner of the project.

The Task table includes the following columns:

Column name	Data type	Notes
TaskId	int	This is a unique identifier for a task.
TaskName	varchar(100)	A nonclustered index exists for this column.
ParentTaskId	int	Each task may or may not have a parent task.
ProjectId	int	A null value indicates the task is not assigned to a specific project.
StartTime	datetime2(7)	
EndTime	datetime2(7)	A null value indicates the task is not completed yet.
UserId	int	Identifies the owner of the task.

You plan to run the following query to update tasks that are not yet started:

UPDATE Task SET StartTime = GETDATE() WHERE StartTime IS NULL

You need to return the total count of tasks that are impacted by this UPDATE operation, but are not associated with a project.

What set of Transact-SQL statements should you run?

A.
```
DECLARE @startedTasks TABLE(ProjectId int)
UPDATE Task SET StartTime = GETDATE() OUTPUT deleted.ProjectId INTO @startedTasks WHERE StartTime is NULL
SELECT COUNT(*) FROM @startedTasks WHERE ProjectId IS NOT NULL
```

B.
```
DECLARE @startedTasks TABLE(TaskId int, ProjectId int)
UPDATE Task SET StartTime = GETDATE() OUTPUT deleted.TaskId, deleted.ProjectId INTO @startedTasks
WHERE StartTime is NULL
SELECT COUNT(*) FROM @startedTasks WHERE ProjectId IS NULL
```

C.
```
DECLARE @startedTasks TABLE(TaskId int)
UPDATE Task SET StartTime = GETDATE() OUTPUT inserted.TaskId, INTO @startedTasks WHERE StartTime is NULL
SELECT COUNT(*) FROM @startedTasks WHERE TaskId IS NOT NULL
```

D.
```
DECLARE @startedTasks TABLE(TaskId int)
UPDATE Task SET StartTime = GETDATE() OUTPUT deleted.TaskId, INTO @startedTasks WHERE StartTime is NULL
SELECT COUNT(*) FROM @startedTasks WHERE TaskId IS NOT NULL
```

A. Option A
B. Option B
C. Option C
D. Option D

Answer: B

Explanation:

The WHERE clause of the third line should be WHERE ProjectID IS NULL, as we want to count the tasks that are not associated with a project.