CHIJEN CHIEN
9/17/21

ANTRA, INC SQL HOMEWORK

28. Write a short essay talking about your understanding of transactions, locks and isolation levels.

Generally, a transaction represents any change within a DBMS against a database. The changes can consist of a single read, write, delete, or update operations or a combination of those. ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties that guarantee the database transactions are processed reliably. Atomicity means all operations in a transaction succeed or every operation is rolled back. Consistency stands for the database is structurally sound on the completion of a transaction. Isolation implies that transactions do not contend with one another. Durability exemplifies the results of applying a transaction are permanent, even in the presence of failures.

SQL Server can operate three different transactions modes: Autocommit Transaction mode, Implicit transaction mode, and Explicit transaction mode. Autocommit transaction is a default mode, and each T-SQL statement is evaluated as a transaction, and they are committed or rolled back according to their results in this mode. If we run a query without mentioning the BEGIN TRAN, it will be considered an implicit transition. On the other hand, if we run a query that starts with BEGIN TRAN and ends with COMMIT or ROLLBACK, it will be considered an explicit transaction. We can also name a transaction and save it use the SAVE TRANSACTION syntax. It can be a save point, and we can use it to roll back any particular part of the transaction rather than the entire transaction.

Concurrency is a situation that arises in a database due to the transaction process. DBMS concurrency is a problem because accessing data simultaneously by two different users can lead to inconsistent results or invalid behavior. SQL Server provides five different levels of transaction isolation to overcome these Concurrency problems. These five isolation levels work on two major concurrency models: the pessimistic model and the optimistic model. In the pessimistic model of managing concurrent data access, the readers can block writers, and the writers can block readers. On the other hand, in the optimistic model of controlling concurrent data access, the readers cannot block writers, and the writers cannot block readers, but the writer can block another writer.

Pessimistic concurrency control utilizes a system of locks to prevent users from modifying data in a way that affects other users. When the database engine processes a statement, the query processor decides which resources need to be accessed and how they will be used. Based on this information, it then determines what types of locks are required to protect each resource. The acquired locks also depend upon the transaction isolation level setting. In short, locks are there to protect resources. Lock modes include Shared, Update, Exclusive, etc.

Shared locks allow concurrent transactions to read a resource, but no other transaction can modify the resource while the lock is held. When transactions might need to update the resource, we use Update locks. Only one transaction can gain an update lock at a time. If data is to be modified, then that lock is converted to an exclusive lock. To ensure that only one transaction can update data at one time, we use Exclusive locks. And nothing else can access that data unless the read uncommitted isolation level is set.

When we connect to a SQL server database, the application can submit queries with one of five different isolation levels. Read Uncommitted, Read Committed, Repeatable Read, and Serializable come under the pessimistic concurrency model. Snapshot comes under the optimistic concurrency model.

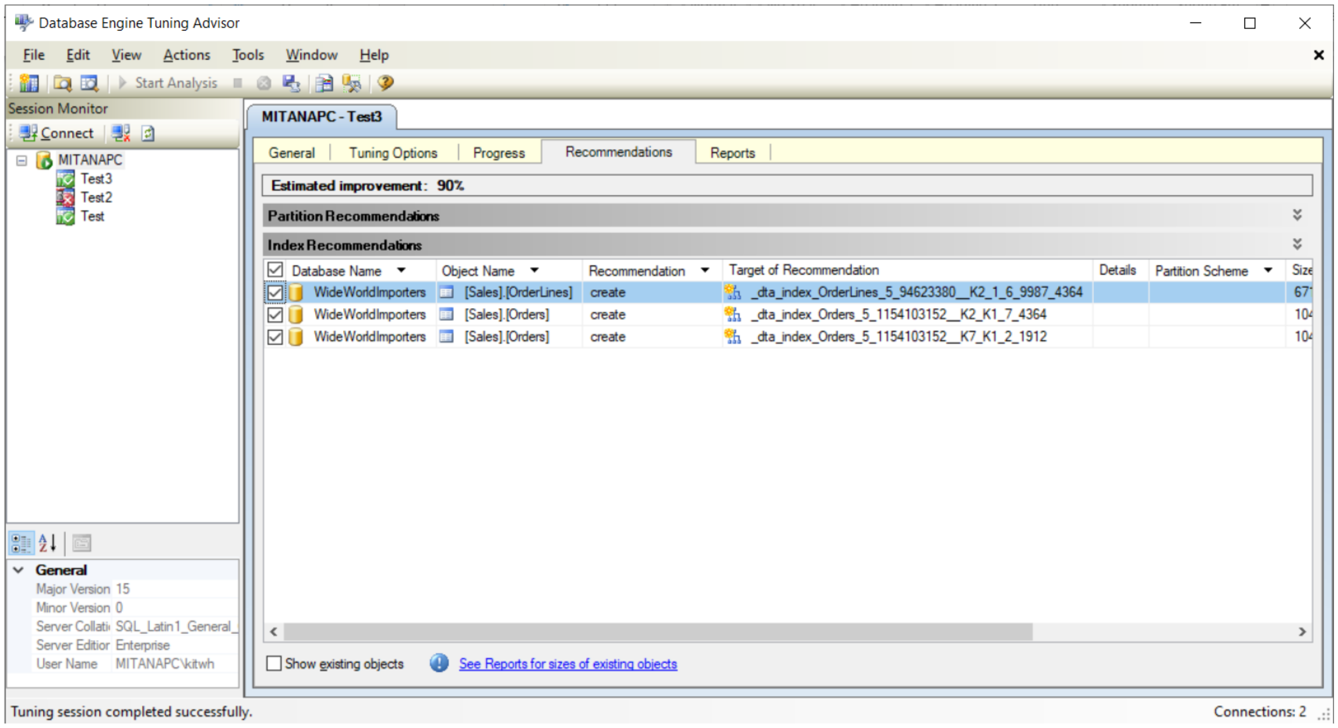
Read Uncommitted is the first level of isolation. One transaction is allowed to read the data that is about to be changed by the commit of another process. Read Committed is the second level of isolation and it is the default level. In this level, if you are reading data then the concurrent transactions that can delete or write data, some work is blocked until the other work is complete. Repeatable Read is the third level of isolation. At this level, the transaction has to wait till another transaction's update or read query is complete. But if there is an insert transaction, it does not wait for anyone. Serializable is the highest level of isolation. In this level of isolation, we can ask any transaction to wait until the current transaction completes.

Snapshot takes a snapshot of the current data and uses it as a copy for the different transactions. Each transaction has its copy of data here. If a user tries to perform a transaction like an update or insert, Snapshot will ask him to re-verify all the operations before the process gets started executing.

29. Write a short essay, plus screenshots talking about performance tuning in SQL Server. Must include Tuning Advisor, Extended Events, DMV, Logs and Execution Plan.

Performance tuning contains a set of processes to optimize the queries to make them run as efficiently as possible. Before doing performance tuning, we have to do performance monitoring first. There are some helpful tools we can use to know how our query performance is. Database Engine Tuning Advisor analyzes databases and gives some recommendations. It helps with a series of things include:
1. Troubleshooting the performance of a specific problem query
2. Tuning a large set of queries across one or more databases
3. Performing exploratory what-if analysis of potential physical design changes
4. Managing storage space.

An example of using Database Engine Tuning Advisor is shown in Fig 1.



Estimated improvement: 90%

✓	Database Name	Object Name	Recommendation	Target of Recommendation	Details	Partition Scheme	Size
✓	WideWorldImporters	[Sales].[OrderLines]	create	_dta_index_OrderLines_5_94623380__K2_1_6_9987_4364			67
✓	WideWorldImporters	[Sales].[Orders]	create	_dta_index_Orders_5_1154103152__K2_K1_7_4364			104
✓	WideWorldImporters	[Sales].[Orders]	create	_dta_index_Orders_5_1154103152__K7_K1_2_1912			104

General
Major Version	15
Minor Version	0
Server Collati	SQL_Latin1_General_
Server Editior	Enterprise
User Name	MITANAPC\kitwh

Tuning session completed successfully.

Figure 1 Results of Database Engine Tuning Advisor

Extended Events provide a set of methods for collecting different events from SQL Server and correlating those different events within a single tool. It's possible to grab:

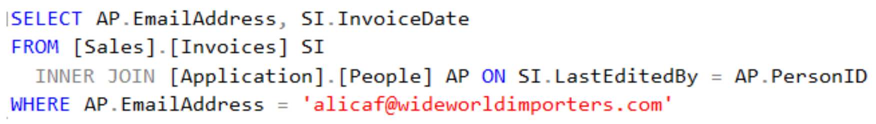
1. Deadlocks + waits

2. Waits + lock graph
3. TempDB spill + query plan

Dynamic Management Views (DMVs) are another great tools to help troubleshoot performance-related issues. Microsoft introduced them in SQL 2005 and added additional DMVs to help troubleshoot issues with each new release. The DMVs act like any other view where you can select data from them, and the DMFs (dynamic management functions) require values to be passed just like any other function.

We can keep logs and data files separated for performance tuning. Log and data files need to be stored on separate drives as the writing and accessing of log files is sequential, whereas writing and accessing of data files is non-sequential. Storing log and data files separately onto other physical drives can enhance the system's performance levels.

SQL Server Execution Plan is a binary representation of the steps that will be followed by the SQL Server Engine to execute the query. We can use it to know is if an index will help and to do performance tuning. Fig 2-4 show the difference before and after creating nonclustered index.



```
SELECT AP.EmailAddress, SI.InvoiceDate
FROM [Sales].[Invoices] SI
  INNER JOIN [Application].[People] AP ON SI.LastEditedBy = AP.PersonID
WHERE AP.EmailAddress = 'alicaf@wideworldimporters.com'
```

Figure 2 Query to do performance tuning

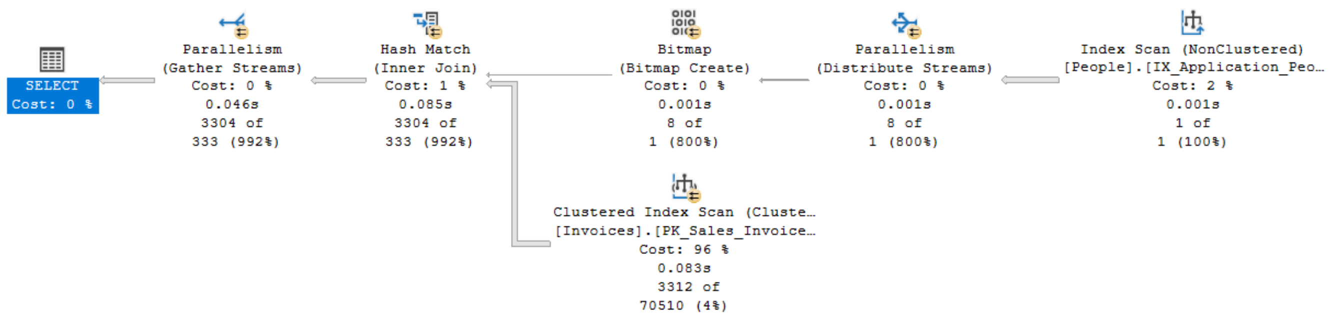


Figure 3 Execution Plan before creating nonclustered index

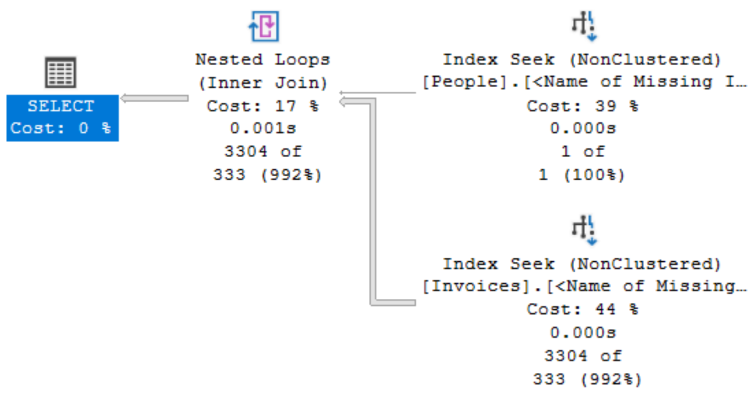
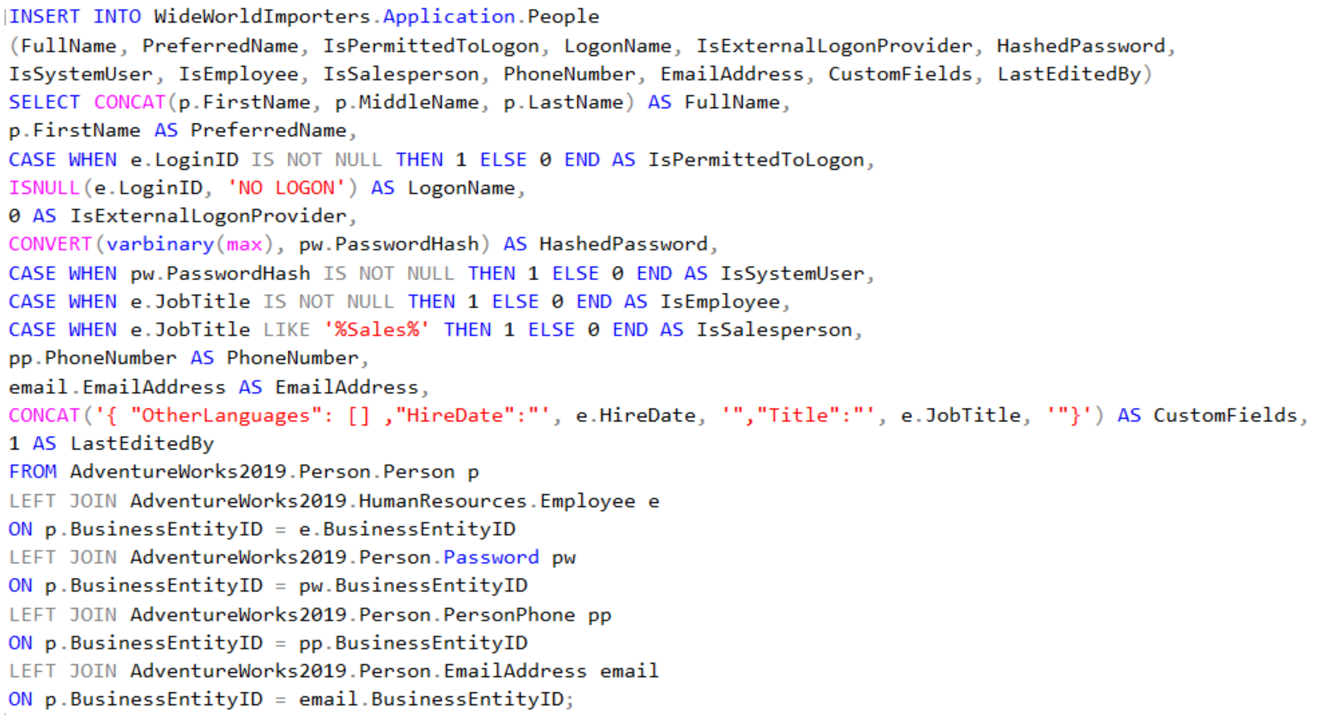


Figure 4 Execution Plan after creating nonclustered index

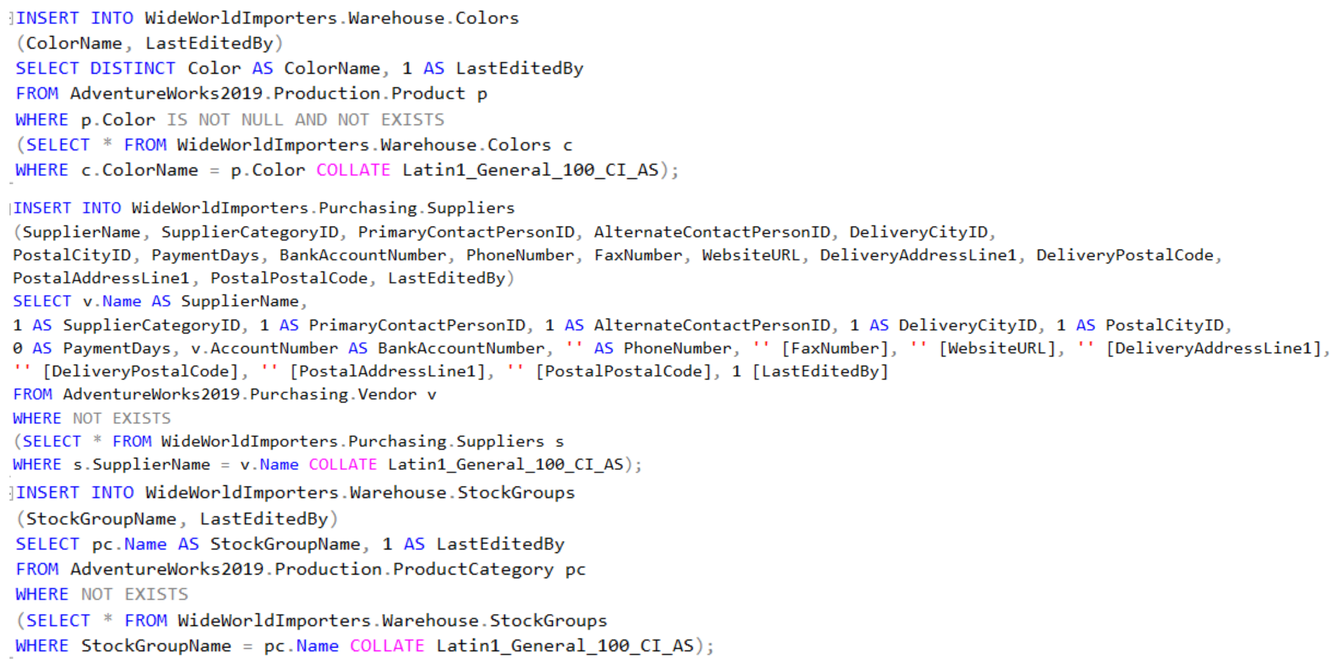
30. Write a short essay talking about a scenario: Good news everyone! We (Wide World Importers) just brought out a small company called “Adventure works”! Now that bike shop is our sub-company. The first thing of all works pending would be to merge the user logon information, person information (including emails, phone numbers) and products (of course, add category, colors) to WWI database. Include screenshot, mapping and query.

First, we loaded the AdventureWorks sample databases from this website: https://docs.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver15&tabs=ssms. Second, we inserted data into "Application. People" in WWI from "Person. Person" in AdventureWorks joined with multiple related tables shown in Fig 5. Third, we inserted data into "Warehouse. Colors", "Purchasing. Suppliers", and “Warehouse.StockGroups” in WWI from "Production. Product" and "Production.ProductCategory" in AdventureWorks shown in Fig 6. Forth, we created a temporal table to save the query results from joined multiple tables related to product information in AdventureWorks and inserted them into "Warehouse.StockItems" in WWI shown in Fig 7. Last, we insert the information to the joined table, “Warehouse.StockItemStockGroups”, in WWI from "Warehouse.StockItems" and “Warehouse.StockGroups” as Fig 8. We met some problems include the different server collations in the two databases, and the duplicated product names that we added the ROW_NUMBER() function to solve.



```sql
INSERT INTO WideWorldImporters.Application.People
(FullName, PreferredName, IsPermittedToLogon, LogonName, IsExternalLogonProvider, HashedPassword,
IsSystemUser, IsEmployee, IsSalesperson, PhoneNumber, EmailAddress, CustomFields, LastEditedBy)
SELECT CONCAT(p.FirstName, p.MiddleName, p.LastName) AS FullName,
p.FirstName AS PreferredName,
CASE WHEN e.LoginID IS NOT NULL THEN 1 ELSE 0 END AS IsPermittedToLogon,
ISNULL(e.LoginID, 'NO LOGON') AS LogonName,
0 AS IsExternalLogonProvider,
CONVERT(varbinary(max), pw.PasswordHash) AS HashedPassword,
CASE WHEN pw.PasswordHash IS NOT NULL THEN 1 ELSE 0 END AS IsSystemUser,
CASE WHEN e.JobTitle IS NOT NULL THEN 1 ELSE 0 END AS IsEmployee,
CASE WHEN e.JobTitle LIKE '%Sales%' THEN 1 ELSE 0 END AS IsSalesperson,
pp.PhoneNumber AS PhoneNumber,
email.EmailAddress AS EmailAddress,
CONCAT('{ "OtherLanguages": [] ,"HireDate":"', e.HireDate, '","Title":"', e.JobTitle, '"}') AS CustomFields,
1 AS LastEditedBy
FROM AdventureWorks2019.Person.Person p
LEFT JOIN AdventureWorks2019.HumanResources.Employee e
ON p.BusinessEntityID = e.BusinessEntityID
LEFT JOIN AdventureWorks2019.Person.Password pw
ON p.BusinessEntityID = pw.BusinessEntityID
LEFT JOIN AdventureWorks2019.Person.PersonPhone pp
ON p.BusinessEntityID = pp.BusinessEntityID
LEFT JOIN AdventureWorks2019.Person.EmailAddress email
ON p.BusinessEntityID = email.BusinessEntityID;
```

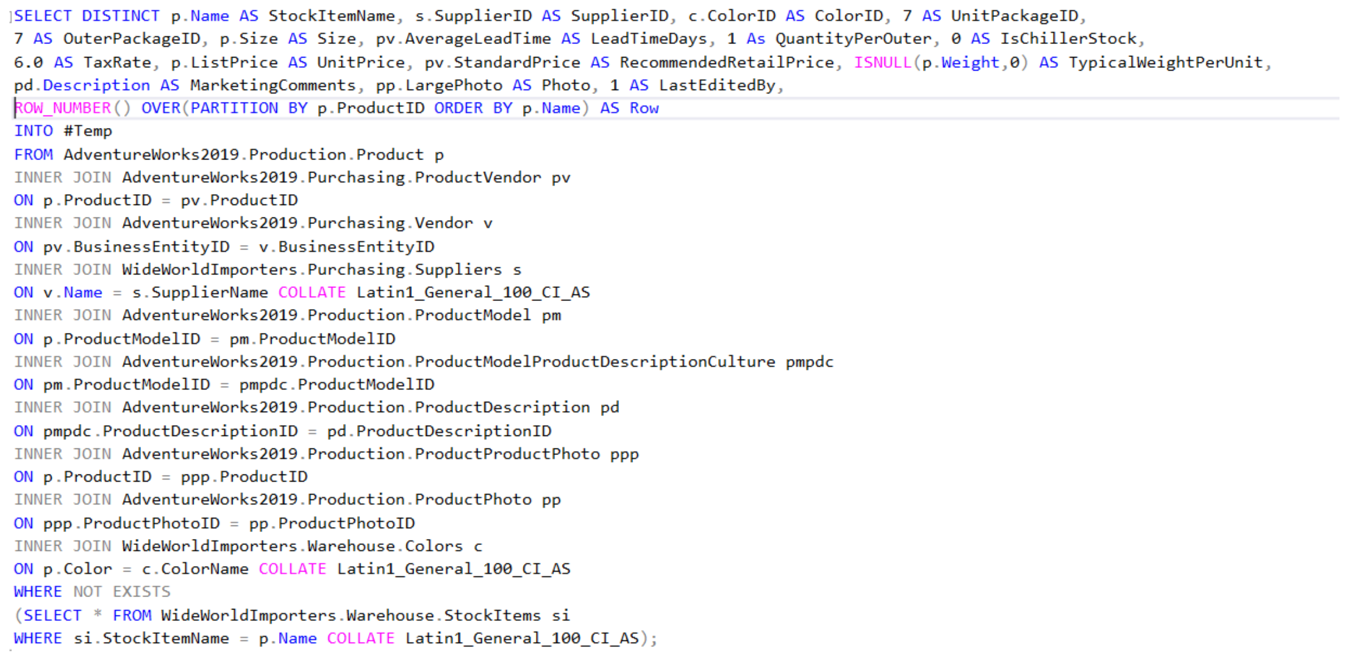
Figure 5   Insert data into People



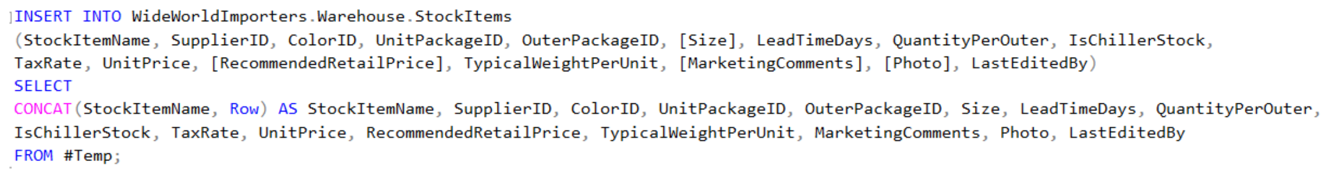
```
INSERT INTO WideWorldImporters.Warehouse.Colors
(ColorName, LastEditedBy)
SELECT DISTINCT Color AS ColorName, 1 AS LastEditedBy
FROM AdventureWorks2019.Production.Product p
WHERE p.Color IS NOT NULL AND NOT EXISTS
(SELECT * FROM WideWorldImporters.Warehouse.Colors c
WHERE c.ColorName = p.Color COLLATE Latin1_General_100_CI_AS);

INSERT INTO WideWorldImporters.Purchasing.Suppliers
(SupplierName, SupplierCategoryID, PrimaryContactPersonID, AlternateContactPersonID, DeliveryCityID,
PostalCityID, PaymentDays, BankAccountNumber, PhoneNumber, FaxNumber, WebsiteURL, DeliveryAddressLine1, DeliveryPostalCode,
PostalAddressLine1, PostalPostalCode, LastEditedBy)
SELECT v.Name AS SupplierName,
1 AS SupplierCategoryID, 1 AS PrimaryContactPersonID, 1 AS AlternateContactPersonID, 1 AS DeliveryCityID, 1 AS PostalCityID,
0 AS PaymentDays, v.AccountNumber AS BankAccountNumber, '' AS PhoneNumber, '' [FaxNumber], '' [WebsiteURL], '' [DeliveryAddressLine1],
'' [DeliveryPostalCode], '' [PostalAddressLine1], '' [PostalPostalCode], 1 [LastEditedBy]
FROM AdventureWorks2019.Purchasing.Vendor v
WHERE NOT EXISTS
(SELECT * FROM WideWorldImporters.Purchasing.Suppliers s
WHERE s.SupplierName = v.Name COLLATE Latin1_General_100_CI_AS);
INSERT INTO WideWorldImporters.Warehouse.StockGroups
(StockGroupName, LastEditedBy)
SELECT pc.Name AS StockGroupName, 1 AS LastEditedBy
FROM AdventureWorks2019.Production.ProductCategory pc
WHERE NOT EXISTS
(SELECT * FROM WideWorldImporters.Warehouse.StockGroups
WHERE StockGroupName = pc.Name COLLATE Latin1_General_100_CI_AS);
```

Figure 6 Insert data into Colors, Suppliers, and StockGroups

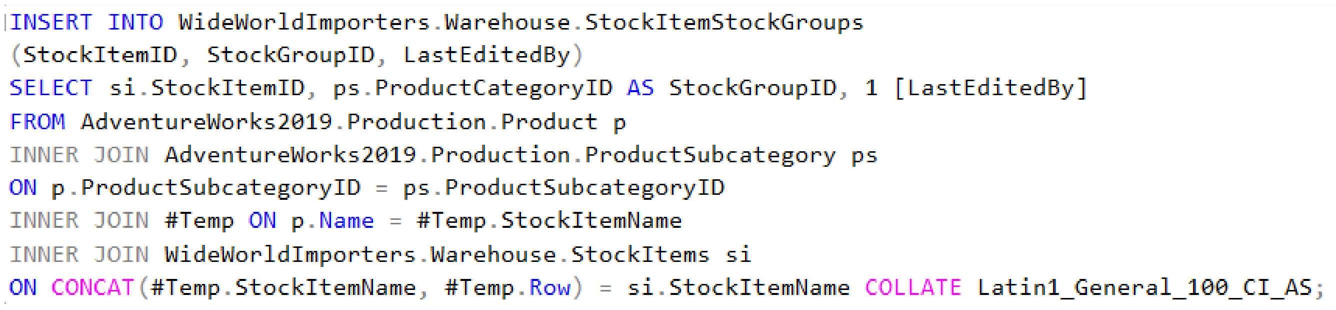


```
SELECT DISTINCT p.Name AS StockItemName, s.SupplierID AS SupplierID, c.ColorID AS ColorID, 7 AS UnitPackageID,
7 AS OuterPackageID, p.Size AS Size, pv.AverageLeadTime AS LeadTimeDays, 1 As QuantityPerOuter, 0 AS IsChillerStock,
6.0 AS TaxRate, p.ListPrice AS UnitPrice, pv.StandardPrice AS RecommendedRetailPrice, ISNULL(p.Weight,0) AS TypicalWeightPerUnit,
pd.Description AS MarketingComments, pp.LargePhoto AS Photo, 1 AS LastEditedBy,
ROW_NUMBER() OVER(PARTITION BY p.ProductID ORDER BY p.Name) AS Row
INTO #Temp
FROM AdventureWorks2019.Production.Product p
INNER JOIN AdventureWorks2019.Purchasing.ProductVendor pv
ON p.ProductID = pv.ProductID
INNER JOIN AdventureWorks2019.Purchasing.Vendor v
ON pv.BusinessEntityID = v.BusinessEntityID
INNER JOIN WideWorldImporters.Purchasing.Suppliers s
ON v.Name = s.SupplierName COLLATE Latin1_General_100_CI_AS
INNER JOIN AdventureWorks2019.Production.ProductModel pm
ON p.ProductModelID = pm.ProductModelID
INNER JOIN AdventureWorks2019.Production.ProductModelProductDescriptionCulture pmpdc
ON pm.ProductModelID = pmpdc.ProductModelID
INNER JOIN AdventureWorks2019.Production.ProductDescription pd
ON pmpdc.ProductDescriptionID = pd.ProductDescriptionID
INNER JOIN AdventureWorks2019.Production.ProductProductPhoto ppp
ON p.ProductID = ppp.ProductID
INNER JOIN AdventureWorks2019.Production.ProductPhoto pp
ON ppp.ProductPhotoID = pp.ProductPhotoID
INNER JOIN WideWorldImporters.Warehouse.Colors c
ON p.Color = c.ColorName COLLATE Latin1_General_100_CI_AS
WHERE NOT EXISTS
(SELECT * FROM WideWorldImporters.Warehouse.StockItems si
WHERE si.StockItemName = p.Name COLLATE Latin1_General_100_CI_AS);
```



```
INSERT INTO WideWorldImporters.Warehouse.StockItems
(StockItemName, SupplierID, ColorID, UnitPackageID, OuterPackageID, [Size], LeadTimeDays, QuantityPerOuter, IsChillerStock,
TaxRate, UnitPrice, [RecommendedRetailPrice], TypicalWeightPerUnit, [MarketingComments], [Photo], LastEditedBy)
SELECT
CONCAT(StockItemName, Row) AS StockItemName, SupplierID, ColorID, UnitPackageID, OuterPackageID, Size, LeadTimeDays, QuantityPerOuter,
IsChillerStock, TaxRate, UnitPrice, RecommendedRetailPrice, TypicalWeightPerUnit, MarketingComments, Photo, LastEditedBy
FROM #Temp;
```

Figure 7 Insert data into StockItems



```sql
INSERT INTO WideWorldImporters.Warehouse.StockItemStockGroups
(StockItemID, StockGroupID, LastEditedBy)
SELECT si.StockItemID, ps.ProductCategoryID AS StockGroupID, 1 [LastEditedBy]
FROM AdventureWorks2019.Production.Product p
INNER JOIN AdventureWorks2019.Production.ProductSubcategory ps
ON p.ProductSubcategoryID = ps.ProductSubcategoryID
INNER JOIN #Temp ON p.Name = #Temp.StockItemName
INNER JOIN WideWorldImporters.Warehouse.StockItems si
ON CONCAT(#Temp.StockItemName, #Temp.Row) = si.StockItemName COLLATE Latin1_General_100_CI_AS;
```

Figure 8 Insert data into StockItemStockGroups