SQL Assignments

Donghao Li, Xuan Li

SQL related assignments will be on Wide World Importers Database if not otherwise introduced.

1. List of Persons’ full name, all their fax and phone numbers, as well as the phone number and fax of the company they are working for (if any).

SELECT p.FullName, p.FaxNumber, p.PhoneNumber, o.FaxNumber, o.PhoneNumber

FROM Application.People as p

LEFT JOIN Sales.Customers as o ON

(o.PrimaryContactPersonID=p.PersonID or o.AlternateContactPersonID=p.PersonID);

1. If the customer's primary contact person has the same phone number as the customer’s phone number, list the customer companies.

SELECT o.CustomerName

FROM Application.People as p

LEFT JOIN Sales.Customers as o ON

(o.PrimaryContactPersonID=p.PersonID or o.AlternateContactPersonID=p.PersonID)

where p.FaxNumber=o.FaxNumber;

1. List of customers to whom we made a sale prior to 2016 but no sale since 2016-01-01.

SELECT distinct c.CustomerName

FROM Sales.Customers as c

LEFT JOIN Sales.Orders as o ON c.CustomerID=o.CustomerID

where o.OrderDate < '2016-01-01'

Except (SELECT c1.CustomerName

FROM Sales.Customers as c1

LEFT JOIN Sales.Orders as o1 ON c1.CustomerID=o1.CustomerID

where o1.OrderDate > '2016-01-01');

1. List of Stock Items and total quantity for each stock item in Purchase Orders in Year 2013.

Select s.StockItemName, sum(ol.Quantity) as 'total quantity'

from Warehouse.StockItems as s

join Warehouse.StockItemTransactions as st

on s.StockItemID = st.StockItemID

join sales.OrderLines ol on ol.StockItemID = s.StockItemID

where st.TransactionOccurredWhen like '2013%'

group by s.StockItemName;

1. List of stock items that have at least 10 characters in description.

select distinct si.StockItemName from

Warehouse.StockItems si

join sales.OrderLines ol on ol.StockItemID = si.StockItemID

where len(ol.Description) >= 10;

1. List of stock items that are not sold to the state of Alabama and Georgia in 2014.

SELECT distinct s.StockItemName from Warehouse.StockItems s

WHERE s.StockItemName not in (

SELECT s.StockItemName from Warehouse.StockItems s

join Warehouse.StockItemTransactions st

join Sales.Customers ct

on st.CustomerID = ct.CustomerID

on s.StockItemID = st.StockItemID

join Sales.Customers on st.CustomerID = ct.CustomerID

join Sales.Orders o

on ct.CustomerID = o.CustomerID

join Application.Cities c

on ct.DeliveryCityID = c.CityID

join Application.StateProvinces sp

on c.StateProvinceID = sp.StateProvinceID

where o.OrderDate like '2014%' and (sp.StateProvinceID = 1 or sp.StateProvinceID = 11));

SELECT distinct s.StockItemName from Warehouse.StockItems s

join Warehouse.StockItemTransactions st

join Sales.Customers ct

on st.CustomerID = ct.CustomerID

on s.StockItemID = st.StockItemID

join Sales.Customers on st.CustomerID = ct.CustomerID

join Sales.Orders o

on ct.CustomerID = o.CustomerID

join Application.Cities c

on ct.DeliveryCityID = c.CityID

join Application.StateProvinces sp

on c.StateProvinceID = sp.StateProvinceID

where YEAR(o.OrderDate) = 2014 and (sp.StateProvinceName IN ('Alabama', 'Georgia'));

1. List of States and Avg dates for processing (confirmed delivery date – order date).

select StateProvinceName, avg(temp.q) as 'process time' from

(select sp.StateProvinceName, datediff(day, o.OrderDate, s.ConfirmedDeliveryTime) as q

from sales.Invoices s

join sales.Orders o on s.OrderID = o.OrderID

join sales.Customers c on c.CustomerID = o.CustomerID

join Application.Cities ct on ct.CityID = c.PostalCityID

join Application.StateProvinces sp on ct.StateProvinceID = sp.StateProvinceID

group by sp.StateProvinceName, s.ConfirmedDeliveryTime, o.OrderDate) as temp

group by StateProvinceName

1. List of States and Avg dates for processing (confirmed delivery date – order date) by month.

select sp.StateProvinceName, avg(datediff(day, o.OrderDate, s.ConfirmedDeliveryTime)) as 'Processing time', month(OrderDate) as mon

from sales.Invoices s

join sales.Orders o on s.OrderID = o.OrderID

join sales.Customers c on c.CustomerID = o.CustomerID

join Application.Cities ct on ct.CityID = c.PostalCityID

join Application.StateProvinces sp on ct.StateProvinceID = sp.StateProvinceID

group by sp.StateProvinceName,month(OrderDate)

order by sp.StateProvinceName, mon

1. List of StockItems that the company purchased more than sold in the year of 2015.

select distinct StockItemName from Warehouse.StockItems s

join Warehouse.StockItemTransactions st on s.StockItemID = st.StockItemID

join Purchasing.PurchaseOrderLines pol on st.StockItemID = pol.StockItemID

join Purchasing.PurchaseOrders po on pol.PurchaseOrderID = po.PurchaseOrderID

join sales.OrderLines ol on ol.StockItemID = s.StockItemID

join sales.InvoiceLines il on il.StockItemID = s.StockItemID

where po.OrderDate like '2015%'

group by StockItemName, CustomerID, ol.Quantity, il.Quantity

having ol.Quantity > il.Quantity

1. List of Customers and their phone number, together with the primary contact person’s name, to whom we did not sell more than 10 mugs (search by name) in the year 2016.

select distinct PhoneNumber, PrimaryContactPersonID from

sales.Customers c

join sales.Orders o on o.CustomerID = c.CustomerID

join sales.OrderLines ol on ol.OrderID = o.OrderID

join warehouse.StockItems si on si.StockItemID = ol.StockItemID

where o.OrderDate like'2016%' and si.StockItemName like '%mug%'

group by PhoneNumber, PrimaryContactPersonID

having count(si.StockItemID) < 10;

1. List all the cities that were updated after 2015-01-01.

select c.CityName from

Application.Cities c

where c.ValidFrom > '2015-01-01';

1. List all the Order Detail (Stock Item name, delivery address, delivery state, city, country, customer name, customer contact person name, customer phone, quantity) for the date of 2014-07-01. Info should be relevant to that date.

select \* from Sales.Orders

where OrderDate = '2014-07-01'

1. List of stock item groups and total quantity purchased, total quantity sold, and the remaining stock quantity (quantity purchased – quantity sold)

select StockItemStockGroupID, sum(ol.Quantity) as 'total quantity purchased', sum(il.Quantity) as 'total quantity sold', (sum(ol.Quantity) - sum(il.Quantity)) as 'remaining stock quantity'

from Warehouse.StockItemStockGroups sisg

join sales.OrderLines ol on sisg.StockItemID = ol.StockItemID

join sales.InvoiceLines il on il.StockItemID = ol.StockItemID

group by StockItemStockGroupID

1. List of Cities in the US and the stock item that the city got the most deliveries in 2016. If the city did not purchase any stock items in 2016, print “No Sales”.

select CityName, StockItemName, Max(tmp.num) as 'Most Delivery'

from

(select CityName, StockItemName, count(ol.OrderID) as num

from Sales.OrderLines ol

join Warehouse.StockItems s on ol.StockItemID = s.StockItemID

join Sales.Orders o on o.OrderID = ol.OrderID

join Sales.Customers c on c.CustomerID = o.CustomerID

join Application.Cities ct on ct.CityID = c.DeliveryCityID

group by CityName, StockItemName) as tmp

group by CityName, StockItemName

1. List any orders that had more than one delivery attempt (located in invoice table).

SELECT i.CustomerID from sales.Invoices i

cross apply openjson(i.returneddeliverydata) with(event varchar(200) '$.event')

where event='DeliveryAttempt'

group by i.OrderID, i.CustomerID

having count(event) > 1;

1. List all stock items that are manufactured in China. (Country of Manufacture)

select StockItemName, JSON\_VALUE(s.CustomFields, '$.CountryOfManufacture') as Country

from Warehouse.StockItems s

where JSON\_VALUE(s.CustomFields, '$.CountryOfManufacture') = 'China'

group by StockItemName, JSON\_VALUE(s.CustomFields, '$.CountryOfManufacture');

1. Total quantity of stock items sold in 2015, group by country of manufacturing.

select distinct JSON\_VALUE(s.CustomFields, '$.CountryOfManufacture') as Country, count(o.OrderID) as 'Total Sold'

from Warehouse.StockItems s

join Sales.OrderLines ol on ol.StockItemID = s.StockItemID

join Sales.Orders o on o.OrderID = ol.OrderID

where o.OrderDate like '2015%'

group by StockItemName, JSON\_VALUE(s.CustomFields, '$.CountryOfManufacture');

1. Create a view that shows the total quantity of stock items of each stock group sold (in orders) by year 2013-2017. [Stock Group Name, 2013, 2014, 2015, 2016, 2017]

create view sales.StockItemSold

as

select year(o.OrderDate) as y, StockGroupName, count(Quantity) as Sold

from Warehouse.StockGroups sg

join Warehouse.StockItemStockGroups sisg on sg.StockGroupID = sisg.StockGroupID

join Warehouse.StockItems si on si.StockItemID = sisg.StockItemID

join Sales.OrderLines ol on ol.StockItemID = si.StockItemID

join sales.Orders o on o.OrderID = ol.OrderID

where year(o.OrderDate) in (2013, 2014, 2015, 2016, 2017)

group by year(o.OrderDate), StockGroupName;

select \* from sales.StockItemSold

order by y

1. Create a view that shows the total quantity of stock items of each stock group sold (in orders) by year 2013-2017. [Year, Stock Group Name1, Stock Group Name2, Stock Group Name3, … , Stock Group Name10]

create view soldquantitybygroup as

select \* from

(

select year(o.OrderDate) as year, sum(ol.quantity) as quantity, sg.StockGroupName as stockgroupname

from Warehouse.StockItems s

join sales.OrderLines ol on ol.StockItemID=s.StockItemID

join Warehouse.StockItemStockGroups sisg on sisg.StockItemID=s.StockItemID

join Warehouse.StockGroups sg on sg.StockGroupID=sisg.StockGroupID

join sales.orders o on o.orderID = ol.OrderID

WHERE year(o.OrderDate) between '2013' and '2017'

GROUP BY year(o.OrderDate), s.StockItemName, sg.StockGroupName) as sourcetable

pivot(sum(quantity) for stockgroupname in ([Airline Novelties],

[Clothing],

[Computing Novelties],

[Furry Footwear],

[Mugs],

[Novelty Items],

[Packaging Materials],

[Toys],

[T-Shirts],

[USB Novelties])) as pv1;

1. Create a function, input: order id; return: total of that order. List invoices and use that function to attach the order total to the other fields of invoices.

create function func(@OrderID int)

returns table

as

return(

select o.OrderID, sum(Quantity) as Total

from Sales.Orders o

join Sales.OrderLines ol on o.OrderID = ol.OrderID

join Sales.Invoices i on i.OrderID = o.OrderID

join Sales.InvoiceLines il on il.InvoiceID = i.InvoiceID

where o.OrderID = @OrderID

group by o.OrderID)

select \* from sales.Invoices

cross apply dbo.func(OrderID);

1. Create a new table called ods.Orders. Create a stored procedure, with proper error handling and transactions, that input is a date; when executed, it would find orders of that day, calculate order total, and save the information (order id, order date, order total, customer id) into the new table. If a given date is already existing in the new table, throw an error and roll back. Execute the stored procedure 5 times using different dates.

create table [ods.Orders](

OrderID int not null primary key,

OrderDate date,

OrderTotal int,

CustomerID int);

create procedure orderSold(@OrderDate date)

as

set nocount on

begin try

begin transaction

insert into[ods.Orders]

select o.OrderID, OrderDate, sum(Quantity), o.CustomerID

from sales.OrderLines ol

join sales.Orders o on o.OrderID = ol.OrderID

where o.OrderDate = @OrderDate

group by o.OrderID, OrderDate, o.CustomerID

commit transaction

end try

begin catch

print('error')

rollback transaction

end catch

return

1. Create a new table called ods.StockItem. It has following columns: [StockItemID], [StockItemName] ,[SupplierID] ,[ColorID] ,[UnitPackageID] ,[OuterPackageID] ,[Brand] ,[Size] ,[LeadTimeDays] ,[QuantityPerOuter] ,[IsChillerStock] ,[Barcode] ,[TaxRate] ,[UnitPrice],[RecommendedRetailPrice] ,[TypicalWeightPerUnit] ,[MarketingComments] ,[InternalComments], [CountryOfManufacture], [Range], [Shelflife]. Migrate all the data in the original stock item table.

select [StockItemID], [StockItemName] ,[SupplierID] ,[ColorID] ,[UnitPackageID] ,

[OuterPackageID] ,[Brand] ,[Size] ,[LeadTimeDays] ,[QuantityPerOuter] ,[IsChillerStock] ,[Barcode] ,[TaxRate]

,[UnitPrice],[RecommendedRetailPrice] ,[TypicalWeightPerUnit] ,[MarketingComments] ,[InternalComments],

JSON\_VALUE(CustomFields, '$.CountryOfManufacture') as [CountryofManufacture]

into[ods.StockItem]

from Warehouse.StockItems

alter table [ods.StockItem]

add [Range] varchar, [Shelflife] varchar;

1. Rewrite your stored procedure in (21). Now with a given date, it should wipe out all the order data prior to the input date and load the order data that was placed in the next 7 days following the input date.

create procedure proc2(@OrderDate date)

as

begin

delete from [ods.Orders]

where OrderDate < @OrderDate

select \* from [ods.Orders]

where OrderDate between @OrderDate and dateadd(day, 7, @OrderDate)

end

1. Consider the JSON file:

{

"PurchaseOrders":[

{

"StockItemName":"Panzer Video Game",

"Supplier":"7",

"UnitPackageId":"1",

"OuterPackageId":[

6,

7

],

"Brand":"EA Sports",

"LeadTimeDays":"5",

"QuantityPerOuter":"1",

"TaxRate":"6",

"UnitPrice":"59.99",

"RecommendedRetailPrice":"69.99",

"TypicalWeightPerUnit":"0.5",

"CountryOfManufacture":"Canada",

"Range":"Adult",

"OrderDate":"2018-01-01",

"DeliveryMethod":"Post",

"ExpectedDeliveryDate":"2018-02-02",

"SupplierReference":"WWI2308"

},

{

"StockItemName":"Panzer Video Game",

"Supplier":"5",

"UnitPackageId":"1",

"OuterPackageId":"7",

"Brand":"EA Sports",

"LeadTimeDays":"5",

"QuantityPerOuter":"1",

"TaxRate":"6",

"UnitPrice":"59.99",

"RecommendedRetailPrice":"69.99",

"TypicalWeightPerUnit":"0.5",

"CountryOfManufacture":"Canada",

"Range":"Adult",

"OrderDate":"2018-01-025",

"DeliveryMethod":"Post",

"ExpectedDeliveryDate":"2018-02-02",

"SupplierReference":"269622390"

}

]

}

Looks like that it is our missed purchase orders. Migrate these data into Stock Item, Purchase Order and Purchase Order Lines tables. Of course, save the script.

DECLARE @json NVARCHAR(MAX) =

'{"PurchaseOrders":

[{"StockItemName":"Panzer Video Game",

"Supplier":"7",

"UnitPackageId":"1",

"OuterPackageId":"6",

"Brand":"EA Sports",

"LeadTimeDays":"5",

"QuantityPerOuter":"1",

"TaxRate":"6",

"UnitPrice":"59.99",

"RecommendedRetailPrice":"69.99",

"TypicalWeightPerUnit":"0.5",

"CountryOfManufacture":"Canada",

"Range":"Adult",

"OrderDate":"2018-01-01",

"DeliveryMethod":"Post",

"ExpectedDeliveryDate":"2018-02-02",

"SupplierReference":"WWI2308"},

{"StockItemName":"Panzer Video Game 2",

"Supplier":"5",

"UnitPackageId":"1",

"OuterPackageId":"7",

"Brand":"EA Sports",

"LeadTimeDays":"5",

"QuantityPerOuter":"1",

"TaxRate":"6",

"UnitPrice":"59.99",

"RecommendedRetailPrice":"69.99",

"TypicalWeightPerUnit":"0.5",

"CountryOfManufacture":"Canada",

"Range":"Adult",

"OrderDate":"2018-01-25",

"DeliveryMethod":"Post",

"ExpectedDeliveryDate":"2018-02-02",

"SupplierReference":"269622390"}]}'

IF OBJECT\_ID(N'tempdb..#JsonTempTable') IS NOT NULL

BEGIN

DROP TABLE #JsonTempTable

END

SELECT \*

INTO #JsonTempTable

FROM OPENJSON(@json)

WITH (PurchaseOrders nvarchar(max) '$.PurchaseOrders' AS JSON) as [t1]

CROSS APPLY OPENJSON(t1.PurchaseOrders)

WITH (StockItemName nvarchar(100), Supplier int, UnitPackageId int, OuterPackageId nvarchar(50), Brand nvarchar(50), LeadTimeDays int,

QuantityPerOuter int, TaxRate decimal(18,3), UnitPrice decimal(18,2), RecommendedRetailPrice decimal(18,2), TypicalWeightPerUnit decimal(18,3),

CountryOfManufacture nvarchar(100), Range nvarchar(100), OrderDate date, DeliveryMethod nvarchar(50), ExpectedDeliveryDate date,

SupplierReference nvarchar(20)) as [t2]

INSERT INTO WideWorldImporters.Warehouse.StockItems(StockItemName, SupplierID, ColorID, UnitPackageID, OuterPackageID, Brand, Size,

LeadTimeDays, QuantityPerOuter, IsChillerStock, Barcode, TaxRate, UnitPrice, RecommendedRetailPrice, TypicalWeightPerUnit, MarketingComments,

InternalComments, Photo, CustomFields, LastEditedBy)

SELECT StockItemName, Supplier [SupplierID], NULL [ColorID], UnitPackageId [UnitPackageID], OuterPackageId [OuterPackageID],

Brand, NULL [Size], LeadTimeDays, QuantityPerOuter, 0 [IsChillerStock], NULL [Barcode], TaxRate, UnitPrice,

RecommendedRetailPrice, TypicalWeightPerUnit, NULL [MarketingComments], NULL [InternalComments], NULL [Photo],

CONCAT('{ "CountryOfManufacture": "', CountryOfManufacture, '", "Tags": [], "Range": "', Range, '" }') [CustomFields], 1 [LastEditedBy]

FROM

(SELECT \*, ROW\_NUMBER() OVER(ORDER BY OrderDate) [RowNum]

FROM #JsonTempTable) [t1]

INSERT INTO WideWorldImporters.Purchasing.PurchaseOrders(SupplierID, OrderDate, DeliveryMethodID, ContactPersonID, ExpectedDeliveryDate, SupplierReference,

IsOrderFinalized, Comments, InternalComments, LastEditedBy, LastEditedWhen)

SELECT Supplier [SupplierID], OrderDate, dm.DeliveryMethodID [DeliveryMethodID], 1 [ContactPersonID], ExpectedDeliveryDate, SupplierReference,

1 [IsOrderFinalized], NULL [Comments], NULL [InternalComments], 1 [LastEditedBy], GETDATE() [LastEditedWhen]

FROM

(SELECT \*, ROW\_NUMBER() OVER(ORDER BY OrderDate) [RowNum]

FROM #JsonTempTable) [t1] JOIN WideWorldImporters.Application.DeliveryMethods dm ON t1.DeliveryMethod = dm.DeliveryMethodName COLLATE database\_default

INSERT INTO WideWorldImporters.Purchasing.PurchaseOrderLines

SELECT RowNum+(SELECT MAX(PurchaseOrderLineID) FROM WideWorldImporters.Purchasing.PurchaseOrderLines) [PurchaseOrderLineID],

RowNum+(SELECT MAX(PurchaseOrderID) FROM WideWorldImporters.Purchasing.PurchaseOrders)-(SELECT COUNT(\*) FROM #JsonTempTable) [PurchaseOrderID],

RowNum+(SELECT MAX(StockItemID) FROM WideWorldImporters.Warehouse.StockItems)-(SELECT COUNT(\*) FROM #JsonTempTable) [StockItemID],

1 [OrderedOuters], '' [Description], 1 [ReceivedOuters], UnitPackageId [PackageTypeID],

UnitPrice [ExpectedUnitPricePerOuter], NULL [LastReceiptDate], 1 [IsOrderLineFinalized], 1 [LastEditedBy], GETDATE() [LastEditedWhen]

FROM

(SELECT \*, ROW\_NUMBER() OVER(ORDER BY OrderDate) [RowNum]

FROM #JsonTempTable) [t1]

1. Revisit your answer in (19). Convert the result in JSON string and save it to the server using TSQL FOR JSON PATH.

select \* from

(

select year(o.OrderDate) as year, sum(ol.quantity) as quantity, sg.StockGroupName as stockgroupname

from Warehouse.StockItems s

join sales.OrderLines ol on ol.StockItemID=s.StockItemID

join Warehouse.StockItemStockGroups sisg on sisg.StockItemID=s.StockItemID

join Warehouse.StockGroups sg on sg.StockGroupID=sisg.StockGroupID

join sales.orders o on o.orderID = ol.OrderID

WHERE year(o.OrderDate) between '2013' and '2017'

GROUP BY year(o.OrderDate), s.StockItemName, sg.StockGroupName) as sourcetable

pivot(sum(quantity) for stockgroupname in ([Airline Novelties],

[Clothing],

[Computing Novelties],

[Furry Footwear],

[Mugs],

[Novelty Items],

[Packaging Materials],

[Toys],

[T-Shirts],

[USB Novelties])) as pv1

for json path;

1. Revisit your answer in (19). Convert the result into an XML string and save it to the server using TSQL FOR XML PATH.

declare @cols as nvarchar(max)

select @cols = coalesce(@cols +',', '') + QUOTENAME(stockgroupname) from (select distinct stockgroupname from Warehouse.StockGroups) a

declare @query as nvarchar(max)

set @query = 'create view stocksoldbygroup as select year,' +'isnull([Airline Novelties], 0) as [AirlineNovelties],[Clothing],

[Computing Novelties] as [ComputingNovelties],[Furry Footwear] as [FurryFootwear],[Mugs],

[Novelty Items] as [NoveltyItems],[Packaging Materials] as [PackagingMaterials],[Toys],[T-Shirts],[USB Novelties] as [USBNovelties]' + 'from

(

select year(o.OrderDate) as year, sum(ol.quantity) as quantity, sg.StockGroupName as stockgroupname

from Warehouse.StockItems s

join sales.OrderLines ol on ol.StockItemID=s.StockItemID

join Warehouse.StockItemStockGroups sisg on sisg.StockItemID=s.StockItemID

join Warehouse.StockGroups sg on sg.StockGroupID=sisg.StockGroupID

join sales.orders o on o.orderID = ol.OrderID

WHERE year(o.OrderDate) in (2013, 2014, 2015, 2016, 2017)

GROUP BY year(o.OrderDate), s.StockItemName, sg.StockGroupName) as sourcetable

pivot(sum(quantity) for stockgroupname in (' + @cols + ')) pv1

for xml path'

exec (@query)

1. Create a new table called ods.ConfirmedDeviveryJson with 3 columns (id, date, value) . Create a stored procedure, input is a date. The logic would load invoice information (all columns) as well as invoice line information (all columns) and forge them into a JSON string and then insert into the new table just created. Then write a query to run the stored procedure for each DATE that customer id 1 got something delivered to him.

drop procedure if exists dbo.storebydate;drop table if exists ods.ConfirmedDeviveryJson;create schema ods;create table ods.ConfirmedDeviveryJson( id int not null primary key identity(1,1), date datetime not null, [value] nvarchar(max) not null);create or alter proc storebydate @date nvarchar(max) as DECLARE @jsonInfo NVARCHAR(MAX)set @jsonInfo = (select i.\*, il.[InvoiceLineID] ,il.[StockItemID] ,il.[Description] ,il.[PackageTypeID] ,il.[Quantity] ,il.[UnitPrice] ,il.[TaxRate] ,il.[TaxAmount] ,il.[LineProfit] ,il.[ExtendedPrice] from sales.Invoices i join sales.InvoiceLines il on i.InvoiceID=il.InvoiceID where i.InvoiceDate = @date for json path); insert into ods.ConfirmedDeviveryJson values (@date, @jsonInfo)godeclare @date\_time datetime;DECLARE CUR\_TEST CURSOR FAST\_FORWARD FOR select distinct i.InvoiceDate from sales.Invoices i where i.CustomerID = 1OPEN CUR\_TESTFETCH NEXT FROM CUR\_TEST INTO @date\_timeWHILE @@FETCH\_STATUS = 0BEGIN exec storebydate @date = @date\_time FETCH NEXT FROM CUR\_TEST INTO @date\_timeENDclose CUR\_TESTDEALLOCATE CUR\_TEST;

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

Transaction is a mechanism, a sequence of operations, which contains a set of database operation commands, which are either all executed or not executed. Therefore, transactions are a group of indivisible logical units of things. When the database is operated concurrently, transactions are used as the smallest control unit. This is especially suitable for data communication systems that are operated by multiple users at the same time such as ticket booking, banks, insurance companies, and securities trading systems.

The locks are needed when multiple users perform concurrent operations on the database at the same time. Some data inconsistency problems will arise. The first one is lost update problem. When two users A and B read the same data and modify it, and the modification result of one user destroys the result of the other modification. Second is dirty read problem. User A modifies the data, and then user B reads the data again, but user A cancels the modification of the data for some reason, and the data is restored to the original value. At this time, the data obtained by B is inconsistent with the data in the database. Third problem is that data cannot be read repeatedly. User A reads the data, and then user B reads the data and modifies it. Currently, user A reads the data again and finds that the two values are inconsistent. The main method of concurrency control is to block. The lock is to prohibit the user from doing certain operations for a period to avoid data inconsistency.

The isolation level determines the behavior and results of concurrent users when reading and writing. Reading generally refers to the Select statement, which uses shared locks by default. Write refers to all statements (Insert, Update, Delete, etc.) that change the content of the database. It requires an exclusive lock. Usually we cannot control whether to add an exclusive lock when writing, but we can specify whether to add a shared lock when reading. Of course, whether to add a shared lock can determine the result of reading, and sometimes it can also affect the result of writing.

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

Tuning Advisor use query optimization tools to analyze the queries in the workload to suggest the best mix of database indexes. It suggests aligned or non-aligned data partitions for the database referenced by the workload. It is recommended to view the index of the database referenced by the workload. It analyzes the effects of the proposed changes, including index usage, query distribution between tables, and query performance in workloads. It is recommended to fine-tune the database method for a small group of question queries.

It allows you to specify advanced options such as disk space constraints to customize recommendations. It provides reports to summarize the effects of implementing recommendations for a given workload. It has imagine alternatives, allowing you to provide possible design options in the form of hypothetical configurations for evaluation by the Tuning Advisor.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text

Description automatically generated

The extended events have the following advantages. Events are triggered synchronously but can be processed synchronously or asynchronously. Any target can consume any event, and any action can be paired with any event, enabling deeper monitoring of the system. It allows you to use Boolean logic to build complex rules. Extended event can use Transact-SQL to fully control the extended event session and monitor performance-critical code without affecting performance.

Graphical user interface, text, application

Description automatically generated

DMV is the metadata of the SQL Server core. Through the analysis of the internal metadata, we can quickly and accurately obtain a lot of information related to the internal SQL Server to perform performance analysis. When the query runs in SQL Server, SQL Server will automatically record the relevant information of this activity and save it in memory. Different types of DMV information have different purposes. For example, related DMVs can be used to diagnose performance, thereby improving performance, or monitoring the operation of the database, or troubleshooting. Because the DMV information is stored in the memory, we don’t need additional operations. We only need to take out the information, perform calculations, statistics, and analysis according to our requirements. The data obtained is very fast and will not be correct. The server is under pressure. In addition, because the DMV is stored by SQL Server itself, and statistical information has been made, the data is closer to the state of the database itself.

Graphical user interface, text, application

Description automatically generated

SQL Server uses logs to record information generated during the startup and operation of the database engine. This information is not necessarily an "error" message. According to the severity of the message, it is divided into three levels, information, warning, and errors. The SQL Server log records the messages in the order in which they occurred. The SQL Server log is used to record the messages generated by the SQL Server engine, and the SQL Server Agent error log is used to record the information generated during the running of the Agent.

Graphical user interface, text, application

Description automatically generated

The execution plan is the execution result of the most effective method of the query optimizer for the T-SQL query request we submitted. The execution plan can tell us how the query is executed. When the database query is troubleshooting, the use of the execution plan is the most important method. There are three ways to display the execution plan, view type, text type, and xml type.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Assignments 30 - 32 are group assignments.

1. 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.

INSERT INTO WideWorldImporters.Application.People(

FullName,

PreferredName,

IsPermittedToLogon,

LogonName,

IsExternalLogonProvider,

HashedPassword,

IsSystemUser,

IsEmployee,

IsSalesperson,

UserPreferences,

PhoneNumber,

FaxNumber,

EmailAddress,

Photo,

CustomFields,

LastEditedBy)

SELECT CONCAT(p.FirstName, ' ', p.MiddleName, ' ', p.LastName) [FullName],

CONCAT(p.FirstName, ' ', p.MiddleName, ' ', p.LastName) [PreferredName],

CASE WHEN e.LoginID IS NOT NULL THEN 1 ELSE 0 END [IsPermittedToLogon], ISNULL(e.LoginID, 'NO LOGON') [LogonName], 0 [IsExternalLogonProvider],

CONVERT(VARBINARY(256), pw.PasswordHash) [HashedPassword], 0 [IsSystemUser], CASE WHEN p.PersonType IN ('SC', 'SP', 'EM') THEN 1 ELSE 0 END [IsEmployee],

CASE WHEN p.PersonType = 'SP' THEN 1 ELSE 0 END [IsSalesperson], NULL [UserPreferences], pp.PhoneNumber [PhoneNumber], NULL [FaxNumber],

ea.EmailAddress [EmailAddress], NULL [Photo], CONCAT('{ "OtherLanguages": [] ,"HireDate":"', e.HireDate, '","Title":"', e.JobTitle, '"}') [CustomFields],

1 [LastEditedBy]

FROM AdventureWorks2019.Person.Person p LEFT JOIN AdventureWorks2019.HumanResources.Employee e ON p.BusinessEntityID = e.BusinessEntityID

JOIN AdventureWorks2019.Person.Password pw ON p.BusinessEntityID = pw.BusinessEntityID

JOIN AdventureWorks2019.Person.EmailAddress ea ON p.BusinessEntityID = ea.BusinessEntityID

JOIN AdventureWorks2019.Person.PersonPhone pp ON p.BusinessEntityID = pp.BusinessEntityID

INSERT INTO WideWorldImporters.Warehouse.StockGroups(StockGroupName, LastEditedBy)

SELECT pc.Name [StockGroupName], 1 [LastEditedBy]

FROM AdventureWorks2019.Production.ProductCategory pc

WHERE NOT EXISTS (SELECT \* FROM WideWorldImporters.Warehouse.StockGroups

WHERE StockGroupName = pc.Name COLLATE SQL\_Latin1\_General\_CP1\_CI\_AS);

INSERT INTO WideWorldImporters.Warehouse.Colors(ColorName, LastEditedBy)

SELECT DISTINCT Color, 1

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 SQL\_Latin1\_General\_CP1\_CI\_AS)

INSERT INTO WideWorldImporters.Purchasing.Suppliers(SupplierName, SupplierCategoryID, PrimaryContactPersonID, AlternateContactPersonID,

DeliveryMethodID, DeliveryCityID, PostalCityID, PaymentDays, BankAccountNumber, PhoneNumber, FaxNumber, WebsiteURL, DeliveryAddressLine1,

DeliveryPostalCode, PostalAddressLine1, PostalPostalCode, LastEditedBy)

SELECT v.Name, 1 [SupplierCategoryID], 1 [PrimaryContactPersonID], 1 [AlternateContactPersonID], 1 [DeliveryMethodID], 1 [DeliveryCityID],

1 [PostalCityID], 0 [PaymentDays], v.AccountNumber [BankAccountNumber], '' [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 SQL\_Latin1\_General\_CP1\_CI\_AS)

SELECT p.Name, s.SupplierID [SupplierID], c.ColorID [ColorID], 7 [UnitPackageID], 7 [OuterPackageID], NULL [Brand], p.Size [Size],

pv.AverageLeadTime [LeadTimeDays], 1 [QuantityPerOuter], 0 [IsChillerStock], NULL [Barcode], 6.0 [TaxRate], p.ListPrice [UnitPrice],

pv.StandardPrice [RecommendedRetailPrice], ISNULL(p.Weight,0) [TypicalWeightPerUnit], pd.Description [MarketingComments], pd.Description [InternalComments],

pp.LargePhoto [Photo], NULL [CustomFields], 1 [LastEditedBy], ROW\_NUMBER() OVER(PARTITION BY p.ProductID ORDER BY p.Name) [Row]

INTO #c

FROM AdventureWorks2019.Production.Product p JOIN AdventureWorks2019.Purchasing.ProductVendor pv ON p.ProductID = pv.ProductID

JOIN AdventureWorks2019.Purchasing.Vendor v ON pv.BusinessEntityID = v.BusinessEntityID

JOIN WideWorldImporters.Purchasing.Suppliers s ON v.Name = s.SupplierName COLLATE SQL\_Latin1\_General\_CP1\_CI\_AS

JOIN AdventureWorks2019.Production.ProductModel pm ON p.ProductModelID = pm.ProductModelID

JOIN AdventureWorks2019.Production.ProductModelProductDescriptionCulture pmpdc ON pm.ProductModelID = pmpdc.ProductModelID

JOIN AdventureWorks2019.Production.ProductDescription pd ON pmpdc.ProductDescriptionID = pd.ProductDescriptionID

JOIN AdventureWorks2019.Production.ProductProductPhoto ppp ON p.ProductID = ppp.ProductID

JOIN AdventureWorks2019.Production.ProductPhoto pp ON ppp.ProductPhotoID = pp.ProductPhotoID

JOIN WideWorldImporters.Warehouse.Colors c ON p.Color = c.ColorName COLLATE SQL\_Latin1\_General\_CP1\_CI\_AS

WHERE NOT EXISTS (SELECT \* FROM WideWorldImporters.Warehouse.StockItems si WHERE si.StockItemName = p.Name COLLATE SQL\_Latin1\_General\_CP1\_CI\_AS)

INSERT INTO WideWorldImporters.Warehouse.StockItems(StockItemName, SupplierID, ColorID, UnitPackageID, OuterPackageID, Brand, Size, LeadTimeDays,

QuantityPerOuter, IsChillerStock, Barcode, TaxRate, UnitPrice, RecommendedRetailPrice, TypicalWeightPerUnit, MarketingComments, InternalComments,

Photo, CustomFields, LastEditedBy)

SELECT Name+CAST(Row AS nvarchar(10)) [StockItemName], SupplierID, ColorID, UnitPackageID, OuterPackageID, Brand, Size, LeadTimeDays,

QuantityPerOuter, IsChillerStock, Barcode, TaxRate, UnitPrice, RecommendedRetailPrice, TypicalWeightPerUnit, MarketingComments, InternalComments,

Photo, CustomFields, LastEditedBy

FROM #c

INSERT INTO WideWorldImporters.Warehouse.StockItemStockGroups(StockItemID, StockGroupID, LastEditedBy)

SELECT si.StockItemID, ps.ProductCategoryID [StockGroupID], 1 [LastEditedBy]

FROM AdventureWorks2019.Production.Product p JOIN AdventureWorks2019.Production.ProductSubcategory ps ON p.ProductSubcategoryID = ps.ProductSubcategoryID

JOIN #c ON p.Name = #c.Name

JOIN WideWorldImporters.Warehouse.StockItems si ON #c.Name+CAST(#c.Row AS nvarchar(10)) = si.StockItemName COLLATE SQL\_Latin1\_General\_CP1\_CI\_AS

A piece of paper with writing

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generatedGraphical user interface, text, application

Description automatically generatedA screenshot of a computer

Description automatically generated

1. Database Design: OLTP db design request for EMS business: when people call 911 for medical emergency, 911 will dispatch UNITs to the given address. A UNIT means a crew on an apparatus (Fire Engine, Ambulance, Medic Ambulance, Helicopter, EMS supervisor). A crew member would have a medical level (EMR, EMT, A-EMT, Medic). All the treatments provided on scene are free. If the patient needs to be transported, that’s where the bill comes in. A bill consists of Units dispatched (Fire Engine and EMS Supervisor are free), crew members provided care (EMRs and EMTs are free), Transported miles from the scene to the hospital (Helicopters have a much higher rate, as you can image) and tax (Tax rate is 6%). Bill should be sent to the patient insurance company first. If there is a deductible, we send the unpaid bill to the patient only. Don’t forget about patient information, medical nature and bill paying status.

Diagram

Description automatically generated

1. Remember the discussion about those two databases from the class, also remember, those data models are not perfect. You can always add new columns (but not alter or drop columns) to any tables. Suggesting adding Ingested DateTime and Surrogate Key columns. Study the Wide World Importers DW. Think the integration schema is the ODS. Come up with a TSQL Stored Procedure driven solution to move the data from WWI database to ODS, and then from the ODS to the fact tables and dimension tables. By the way, WWI DW is a galaxy schema db. Requirements:
   1. Luckly, we only start with 1 fact: Order. Other facts can be ignored for now.
   2. Add a new dimension: Country of Manufacture. It should be given on top of Stock Items.
   3. Write script(s) and stored procedure(s) for the entire ETL from WWI db to DW.

a. create or alter proc wwiorderstaging as insert into WideWorldImportersdw.Integration.Order\_Staging select newid(), tmp1.[City Key], tmp1.[customer key], tmp1.[stock item key], tmp1.OrderDate, tmp1.ExpectedDeliveryDate, dwe.[Employee Key] as salesperson, dwc.[Customer Key] as pickup, o.OrderID, o.BackorderOrderID, ol.Description, p.PackageTypeName, tmp1.Quantity, tmp1.UnitPrice, tmp1.TaxRate, (tmp1.Quantity \* tmp1.UnitPrice) as excluding\_tax\_price, (tmp1.Quantity \* tmp1.UnitPrice) \* (tmp1.TaxRate/100) as taxamount,(tmp1.Quantity \* tmp1.UnitPrice) \* (1+tmp1.TaxRate/100) as total\_price, tmp1.[Lineage Key], tmp1.PostalCityID,tmp1.CustomerID, tmp1.StockItemID, o.SalespersonPersonID, tmp1.PickedByPersonID, o.LastEditedWhen from WideWorldImporters.Sales.orderlines ol left join (select distinct wol.orderlineID, dwct.[City Key],wo.CustomerID, wol.StockItemID, wo.OrderDate, wol.Quantity, wol.UnitPrice,wo.ExpectedDeliveryDate, wo.PickedByPersonID, dwc.[Customer Key] as [customer key], dwp.[Lineage Key], wc.PostalCityID, wol.TaxRate, dwsi.[stock item key]from WideWorldImporters.Sales.OrderLines wol left join WideWorldImporters.Sales.Orders wo on wol.OrderID = wo.orderidleft join WideWorldImporters.Sales.Customers wc on wc.CustomerID = wo.CustomerIDleft join WideWorldImportersDW.Dimension.Customer dwc on dwc.[WWI Customer ID] = wc.CustomerIDleft join WideWorldImporters.Sales.Invoices wi on wi.CustomerID = wc.CustomerIDleft join WideWorldImportersDW.Fact.Sale dwsale on dwsale.[WWI Invoice ID] = wi.InvoiceIDleft join WideWorldImportersDW.Dimension.City dwct on wc.PostalCityID = dwct.[City Key]left join WideWorldImportersDW.Dimension.[Stock Item] dwsi on dwsi.[WWI Stock Item ID] = wol.StockItemIDleft join WideWorldImportersDW.Fact.Purchase dwp on dwp.[WWI Purchase Order ID] = wo.OrderID) as tmp1 on tmp1.OrderlineID=ol.OrderlineIDleft join WideWorldImporters.Sales.Orders o on o.OrderID = ol.OrderIDleft join WideWorldImportersDW.Dimension.Employee dwe on o.SalespersonPersonID = dwe.[Employee Key]left join WideWorldImportersDW.Dimension.Customer dwc on tmp1.PickedByPersonID = dwc.[Customer Key]left join WideWorldImporters.Warehouse.PackageTypes p on ol.PackageTypeID = p.PackageTypeID;create proc stroreorderinsert into WideWorldImportersDW.Fact.[Order] select t.\* from (select [id] ,[City Key] ,[Customer Key] ,[Stock Item Key] ,[Order Date Key] ,[Pickup Date] ,[Salesperson] ,[Pickup] ,[WWIOrderID] ,[WWIBackorderID] ,[Description] ,[Packagetypename] ,[Quantity] ,[UnitPrice] ,[Rate] ,[excluding\_tax\_price] ,[TaxAmount] ,[Total\_price] ,[Lineage Key] from WideWorldImportersDW.integration.order) t where (t.[customer key] is not null) and (t.[Lineage key] is not null)b.use WideWorldImporters;use WideWorldImportersDW;drop table if exists wideworldImportersdw.dimension.country;with cte as (select si.stockitemid stockitemid, sum(ol.Quantity) as total\_quantity, JSON\_VALUE(si.customfields, '$.CountryOfManufacture')as country from wideworldimporters.warehouse.StockItems si join wideworldimporters.sales.OrderLines ol on ol.StockItemID = si.StockItemIDgroup by si.StockItemID, JSON\_VALUE(si.customfields, '$.CountryOfManufacture'))select cte.stockitemid, cte.country, max(cte.total\_quantity) into wideworldImportersdw.dimension.country from cte join(select max(total\_quantity) [maximum quantity], country from cte group by country) as tmp on tmp.country = cte.countrygroup by cte.country, cte.stockitemid