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

A transaction is a block/unit of work that contains one or more activities. In a transaction, either all activities would succeed or no activities would succeed. It acts as a save point for when we want to make changes to the database. It's important for ensuring data integrity and handling database errors. Some common types of transactions include autocommit transaction, explicit transaction, implicit transaction, and save transaction. Autocommit transaction is the default mode for the SQL server; for example, when we try to make changes to the database, if we can execute our statement successfully, this transaction is automatically committed, whereas if the statement violates the constraints of the table, the system will throw an error, and automatically roll back the transaction. Explicit transaction is when we clearly defines a transaction using the command BEGIN TRAN, and can be rolled back or committed using commands ROLLBACK TRAN and COMMIT TRAN. Implicit transaction is a mode that could be manually turned on by writing SET IMPLICIT_TRANSACTIONS ON, and the transaction will begin without the user specifying; however, it requires an explicit commit. Save transaction allows us to create a save point in a transaction using SAVE TRANSACTION, and give us a chance to roll back only part of the transaction.

Locks are relevant to concurrency control and to prevent concurrency issues. Locks are essentially constraints that the system put on the tables will prevent users from making modifications to or simply read the data in a way that affect other users. A deadlock happens on multiple transactions happening at the same time, and it's resolved automatically by letting the more expensive transaction and rolling back the rest. The reason why it's resolved automatically is that a deadlock may cause the computer is essentially stuck in an infinite loop. For example, if two transactions need both Table A and Table B, what happens is that Transaction A will place a lock on Table A, while Transaction B will place a lock on Table B; both transactions have to wait on the other transaction to finish before they could continue, which would result in an infinite waiting. In this case, the system will decide which process needs more resources to run and let it through while rolling back the other transaction.

Isolation levels are also relevant to concurrency control. Common isolation levels include, Read Uncommitted, Read Committed, Repeatable Read, Serializable and Snapshot. Read Uncommitted is the first level of isolation under pessimistic model of concurrency. Users are allowed to read the data that is about to be changed by the commit of another process. For example, if someone is updating a row but this transaction is yet to be committed, another person would be able to read this update; this is also know as the dirty read problem. The second level of isolation levels is Read Committed, which is also the system default. This eliminates the dirty read problem in the first level, and only allows users to see the committed changes. However, it also more time-consuming, because one has to wait until the other to commit the transaction before reading the data. The next level is Repeatable Read. On top of the Read Committed level, Repeatable Read allows the read data to remain the same after the first read, whereas Committed Read does not guarantee that. This feature in Repeatable Read might also lead to a phantom read if one transaction tries to insert a new row but then this transaction is rolled back, the read data before the rollback would still contain the deleted row. The last level under pessimistic concurrency control is Serializable, where we can ask any transactions to wait until the previous one is completed. It is also the most time-consuming. For optimistic concurrency control, we have Snapshot, which avoids most locking. It uses version numbers to avoid any waits. When a reading transaction begin, the committed version of the data as of the time of the reading would be return. The system achieves this by copying committed versions of affected rows when modified to tempdb and give them version numbers.

Acquisition of Adventure Works Announcement

Bill Lan, Lillian Lei, Chloe Wang.

Abstract

Good news everyone! We, Wide World Importers, are happy to announce that we have 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 AdventureWorks user logon information, person information, and products into our database. This is a process that will be a collaborative effort between our Enterprise Data Analytics team and the Data Engineering team over At Adventure Works as Well!

The Data Engineering Team at AdventureWorks has left us a description of their database, storage methods, and detailed documentation of their database. They have even been kind enough to help us merge our databases, leaving their code and documentation so that we can begin to generate new product categories, changes to the way their data is stored, and their customers to become Wide World Importer customers by making an online account!

1 Application.People

Most data in the Application.People are updated except columns IsSystemUser, IsEmployee and IsSalesperson. We plan to do it in the future by checking whether the person is an employee or a customer or a vendor. If the person is an employee, IsSystemUser and IsEmployee are likely to be 1, otherwise 0. IsEmployee will be 1 if AW.BusinessEntityID is in the Employee Table, otherwise 0. Likewise, IsSalesperson will be 1 if AW.BusinessEntityID is in the SalesPerson Table.

```
ALTER TABLE WideWorldImporters.Application.People
ALTER COLUMN IsSystemUser DROP NOT NULL;
ALTER TABLE WideWorldImporters.Application.People
ALTER COLUMN  IsEmployee DROP NOT NULL;
ALTER TABLE WideWorldImporters.Application.People
ALTER COLUMN IsSalesPerson DROP NOT NULL;

WITH TempTable as (
    SELECT * FROM AdventureWorks2019.Person.Person Person
    LEFT JOIN (
    SELECT * FROM AdventureWorks 2019.Person.EmailAddress
    ) AS PEA
    ON Person.BusinessEntityID = PEA.BusinessEntityID
    LEFT JOIN (
SELECT * FROM AdventureWorks2019.Person.PersonPhone
    ) PPP ON Person.BusinessEntityID = PPP.BusinessEntityID
    )
SELECT
    ROW_NUMBER() + 3261 AS PersonID,
    FirstName AS PreferredName,
    FirstName + ‘ ‘ + MiddleName + ‘ ‘ + LastName AS FullName,
    FirstName + ‘ ‘ + LastName AS SearchName,
    EmailAddress AS LogonName,
    PersonPhone AS PhoneNumber,
    1 AS IsPermittedToLogon,
    1 AS IsExternalLogonProvider ,
    NULL AS HashedPassword,
    NULL AS IsSystemUser,
    NULL AS IsEmployee,
    NULL AS IsSalesPerson,
    NULL AS UserPreferences,
    NULL AS FaxNumber,
    EmailAddress AS EmailAddress,
    NULL AS Photo,
    NULL AS CustomFields,
    NULL AS OtherLanguages,
    NULL AS  LastEditedBy,
    GetDate() AS ValidFrom,
    ‘2099/12/31’ AS ValidTo
FROM TempTable

INSERT INTO WideWorldImporters.Application.People (
PersonID, FullName, PreferredName,
SearchName, IsPermittedToLogon, LogonName,
IsExternalLogonProvider, HashedPassword, IsSystemUser,
IsEmployee, IsSalesperson, UserPreferences,
PhoneNumber, FaxNumber, EmailAddress,
Photo, CustomFields, OtherLanguages,
LastEditedBy, ValidFrom, ValidTo
);
```

1.1 Warehouse.Colors

The Warehouse.Colors Table is up-to-date. We migrated new colors in the AW database and created a new ColorID for each new color. We assume that as the data engineer, we’re essentially PersonID 1, so all the values in the LastEditedBy column is set to 1.

```
INSERT INTO WideWorldImporters.Warehouse.Colors
SELECT ROW_NUMBER() OVER()  + 36 AS ColorID,
P.Color AS ColorName, 1 AS LastEditedBy,
GetDate() AS ValidFrom,
CAST('12/31/9999 23:59:59.999' as datetime) AS ValidTo
FROM AdventureWorks2019.Production.Product P
WHERE P.Color NOT IN (SELECT C.ColorName
FROM WideWorldImporters.Warehouse.Colors C)
```

1.2 Warehouse.StockGroups

For the Warehouse.StockGroup, we updated the database with the AW.ProductCategory table.

```
INSERT INTO WideWorldImporters.Warehouse.StockGroups
SELECT ROW_NUMBER() OVER(ORDER BY C.ProductCategoryID) + 10 AS StockGroupID,
C.Name AS StockGroupName,
1 AS LastEditedBy,
GETDATE() AS ValidFrom,
CAST('12/31/9999 23:59:59.999' as datetime) AS ValidTo
FROM AdventureWorks2019.Production.ProductCategory C
WHERE C.Name NOT IN (SELECT SG.StockGroupName
FROM WideWorldImporters.Warehouse.StockGroups SG)
```

1.3 Warehouse.StockItemGroup

For the Warehouse.StockItemStockGroup, we updated the database with the AW.ProductSubcategory table.

```
INSERT INTO WideWorldImporters.Warehouse.StockGroups
SELECT ROW_NUMBER() OVER(ORDER BY C.ProductCategoryID) + 10 AS StockGroupID,
C.Name AS StockGroupName,
1 AS LastEditedBy,
GETDATE() AS ValidFrom,
CAST('12/31/9999 23:59:59.999' as datetime) AS ValidTo
FROM AdventureWorks2019.Production.ProductCategory C
WHERE C.Name NOT IN (SELECT SG.StockGroupName
FROM WideWorldImporters.Warehouse.StockGroups SG)
```

1.4 Warehouse.StockItemStockGroup

For the Warehouse.StockItemStockGroup, we updated the database with the AW.ProductSubcategory table.

```
INSERT INTO WideWorldImporters.StockItemStockGroup
SELECT ROW_NUMBER() OVER(ORDER BY C.Name) + 10 AS StockItemStockGroupID,
P.ProductID + 227 AS StockItemID,
C.ProductCategoryID + 10 as StockGroupID,
1 AS LastEditedBy,
CAST('12/31/9999 23:59:59.997' as datetime) AS ValidTo
FROM AdventureWorks2019.Production.Product P
JOIN AdventureWorks2019.Production.ProductSubcategory SC
ON P.ProductSubcategoryID = SC.ProductSubcategoryID
JOIN AdventureWorks2019.Production.ProductCategory C
ON SC.ProductCategoryID = C.ProductCategoryID
ORDER BY P.ProductID
SELECT
    P AS StockItemID,
    ROW_NUMBER() OVER(ORDER BY PI.Product) + 443 AS StockItemGroupID,
    1 AS LastEditedBy,
    '2022/08/04 AS LstEditedWhen
FROM AdventureWorks2019.Production.Product P
JOIN AdventureWorks2019.Productions.ProductSubcategory S
P.ProductionCategoryID = S.ProductCategoryID
```

1.5 Warehouse.StockItems

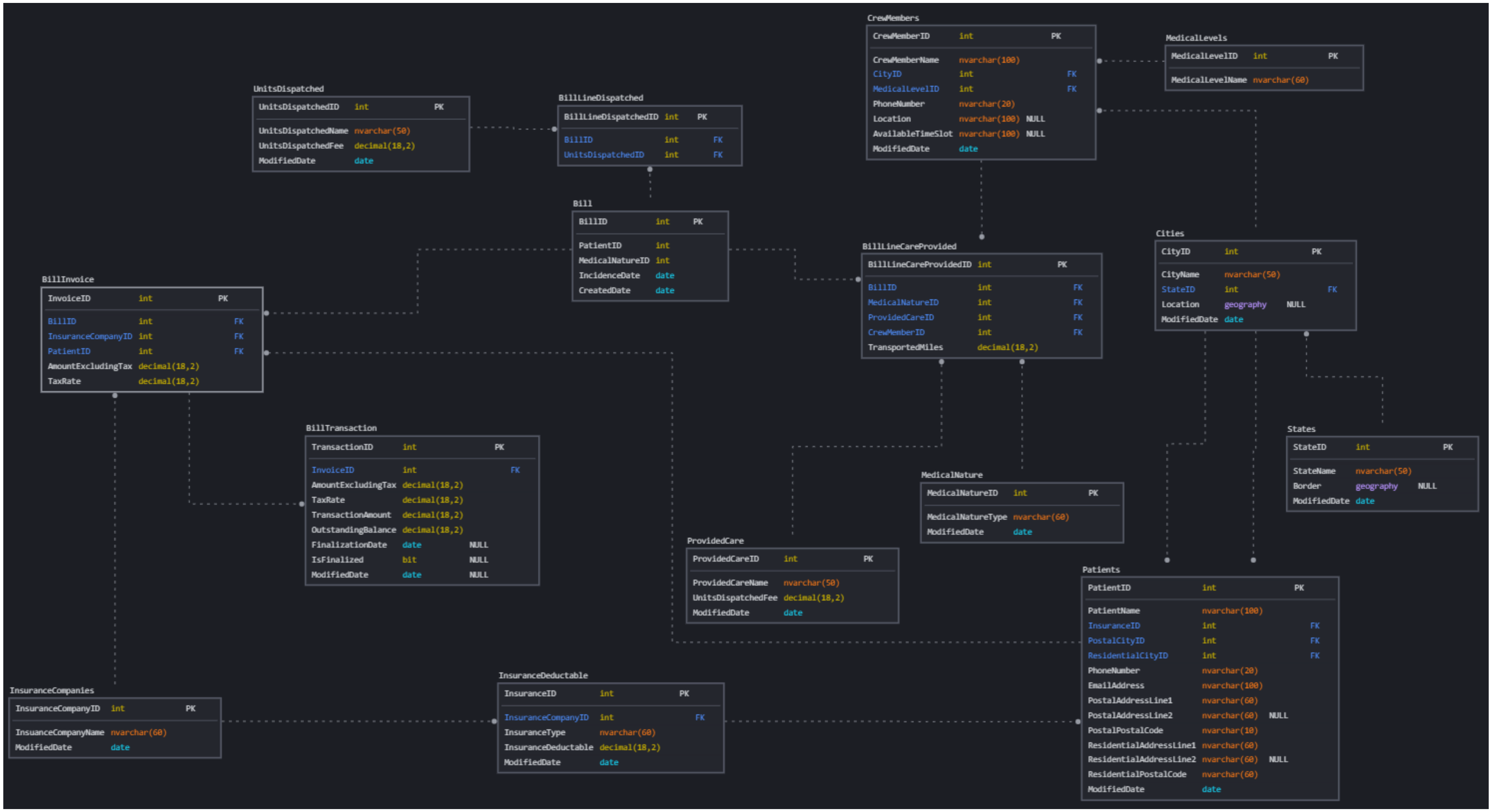
For this stage of data migration, we are only updating the product information for Warehouse.StockItems for the StockItemID, StockItemName, ColorID, as well as ValidFrom and ValidTo. The next stage will be filling in the rest of the columns once we get more information with AdventureWorks.

```
ALTER TABLE WideWorldImporters.Warehouse.StockItems
ALTER COLUMN SupplierID DROP NOT NULL;
ALTER TABLE WideWorldImporters.Warehouse.StockItems
ALTER COLUMN UnitPackageID DROP NOT NULL;
ALTER TABLE WideWorldImporters.Warehouse.StockItems
ALTER COLUMN OuterPackageID DROP NOT NULL;
ALTER TABLE WideWorldImporters.Warehouse.StockItems
ALTER COLUMN LeadTimeDays DROP NOT NULL;
ALTER TABLE WideWorldImporters.Warehouse.StockItems
ALTER COLUMN QuantityPerOuter DROP NOT NULL;
ALTER TABLE WideWorldImporters.Warehouse.StockItems
ALTER COLUMN IsChillerStock DROP NOT NULL;
ALTER TABLE WideWorldImporters.Warehouse.StockItems
ALTER COLUMN TaxRate DROP NOT NULL;
ALTER TABLE WideWorldImporters.Warehouse.StockItems
ALTER COLUMN UnitPrice DROP NOT NULL;
ALTER TABLE WideWorldImporters.Warehouse.StockItems
ALTER COLUMN SearchDetails DROP NOT NULL;

INSERT INTO WideWorldImporters.Warehouse.StockItems (StockItemID,
    StockItemName, ColorID,LastEditedBy, ValidFrom, ValidTo)
SELECT
APP.ProductID + 227 AS StockItemID,
APP.Name AS StockItemName,
WWC.ColorID,
1 AS LastEditedBy,
GetDate() AS ValidFrom,
CAST('12/31/9999 23:59:59.999' as datetime) AS ValidTo
FROM AdventureWorks2019.Production.Product APP
LEFT JOIN WideWorldImporters.Warehouse.Colors WWC
ON APP.Color = WWC.ColorName
```

Question 31. Database Design for EMS Business

Bill Lan, Lillian Lei, Chloe Wang



UnitsDispatched

UnitsDispatchedID	int		PK
UnitsDispatchedName	nvarchar(50)		
UnitsDispatchedFee	decimal(18,2)		
ModifiedDate	date		

BillLineDispatched

BillLineDispatchedID	int		PK
BillID	int		FK
UnitsDispatchedID	int		FK

CrewMembers

CrewMemberID	int		PK
CrewMemberName	nvarchar(100)		
CityID	int		FK
MedicalLevelID	int		FK
PhoneNumber	nvarchar(20)		
Location	nvarchar(100)	NULL	
AvailableTimeSlot	nvarchar(100)	NULL	
ModifiedDate	date		

MedicalLevels

MedicalLevelID	int		PK
MedicalLevelName	nvarchar(60)		

Bill

BillID	int		PK
PatientID	int		
MedicalNatureID	int		
IncidenceDate	date		
CreatedDate	date		

BillLineCareProvided

BillLineCareProvidedID	int		PK
BillID	int		FK
MedicalNatureID	int		FK
ProvidedCareID	int		FK
CrewMemberID	int		FK
TransportedMiles	decimal(18,2)		

Cities

CityID	int		PK
CityName	nvarchar(50)		
StateID	int		FK
Location	geography	NULL	
ModifiedDate	date		

BillInvoice

InvoiceID	int		PK
BillID	int		FK
InsuranceCompanyID	int		FK
PatientID	int		FK
AmountExcludingTax	decimal(18,2)		
TaxRate	decimal(18,2)		

BillTransaction

TransactionID	int		PK
InvoiceID	int		FK
AmountExcludingTax	decimal(18,2)		
TaxRate	decimal(18,2)		
TransactionAmount	decimal(18,2)		
OutstandingBalance	decimal(18,2)		
FinalizationDate	date	NULL	
IsFinalized	bit	NULL	
ModifiedDate	date	NULL	

States

StateID	int		PK
StateName	nvarchar(50)		
Border	geography	NULL	
ModifiedDate	date		

MedicalNature

MedicalNatureID	int		PK
MedicalNatureType	nvarchar(60)		
ModifiedDate	date		

ProvidedCare

ProvidedCareID	int		PK
ProvidedCareName	nvarchar(50)		
UnitsDispatchedFee	decimal(18,2)		
ModifiedDate	date		

Patients

PatientID	int		PK
PatientName	nvarchar(100)		
InsuranceID	int		FK
PostalCityID	int		FK
ResidentialCityID	int		FK
PhoneNumber	nvarchar(20)		
EmailAddress	nvarchar(100)		
PostalAddressLine1	nvarchar(60)		
PostalAddressLine2	nvarchar(60)	NULL	
PostalPostalCode	nvarchar(10)		
ResidentialAddressLine1	nvarchar(60)		
ResidentialAddressLine2	nvarchar(60)	NULL	
ResidentialPostalCode	nvarchar(60)		
ModifiedDate	date		

InsuranceCompanies

InsuranceCompanyID	int		PK
InsuanceCompanyName	nvarchar(60)		
ModifiedDate	date		

InsuranceDeductable

InsuranceID	int		PK
InsuranceCompanyID	int		FK
InsuranceType	nvarchar(60)		
InsuranceDeductable	decimal(18,2)		
ModifiedDate	date		

			PK of	Comments
Bill				
BillID	int, not null, identity(1,1)	PK		
PatientID	int, not null	FK	**Patients**	
MedicalNatureID	int, not null	FK	**MedicalNature**	
IncidenceDate	date, not null			
CreatedDate	date, not null			

BillLineDispatched				
BillLineDispatchedID	int, not null, identity(1,1)	PK		
BillID	int, not null	FK	**Bill**	
UnitsDispatchedID	int, not null	FK	**UnitsDispatched**	

BillLineCareProvided				
BillLineCareProvidedID	int, not null, identity(1,1)	PK		
BillID	int, not null		**Bill**	
CrewMemberID	int, not null	FK	**CrewMembers**	
ProvidedCareID	int, not null	FK	**ProvidedCare**	
TransportedMiles	decimal(18,2), not null			
MedicalNatureID		FK	**MedicalNature**	

BillTransaction				
TransactionID	int, not null, identity(1,1)	PK		
InvoiceID	int, not null	FK	**Invoice**	
AmountExcludingTax	decimal(18,2), not null			
TaxRate	decimal(18,2), not null			
TransactionAmount	decimal(18,2), not null			
OutstandingBalance	decimal(18,2), not null			Calculated
FinalizationDate	date, null			
IsFinalized	bit, not null			IF outstanding balance = 0, then finalized
ModifiedDate	date, not null			

BillInvoice				
InvoiceID	int, not null, identity(1,1)	PK		
BillID	int, not null	FK	**Bill**	
PatientID	int, null	FK	**Patients**	IF NOT NULL, bill to patient
InsuranceCompanyID	int, null	FK	**InsuranceCompanies**	IF NOT NULL, bill to insurance company
AmountExcludingTax	decimal(18,2), not null			
TaxRate	decimal(18,2), not null			

			PK of
CrewMembers			
CrewMemberID	int, not null, identity(1,1)	PK	
CrewMemberName	nvarchar(100), not null		
PhoneNumber	nvarchar(20), not null		
MedicalLevelID	int, not null	FK	**MedicalLevels**
Location	nvarchar(100), null		
CityID	int, not null	FK	**Cities**
AvailableTimeSlot	nvarchar(100), null		
ModifiedDate	date, not null		

Patients			
PatientID	int, not null, identity(1,1)	PK	
PatientName	nvarchar(100), not null		
PhoneNumber	nvarchar(20), not null		
EmailAddress	nvarchar(100), not null		
PostalAddressLine1	nvarchar(60), not null		
PostalAddressLine2	nvarchar(60), null		
PostalPostalCode	nvarchar(10), not null		
PostalCityID	int, not null	FK	**Cities**
ResidentialAddressLine1	nvarchar(60), not null		
ResidentialAddressLine2	nvarchar(60), null		
ResidentialPostalCode	nvarchar(10), not null		
ResidentialCityID	int, not null	FK	**Cities**
InsuranceID	int, not null	FK	**Insurance**
ModifiedDate	date, not null		

MedicalLevels			
MedicalLevelID	int, not null, identity(1,1)	PK	
MedicalLevelName	nvarchar(60), not null		

UnitsDispatched			
UnitsDispatchedID	int, not null, identity(1,1)	PK	
UnitsDispatchedName	nvarchar(50)		
UnitsDispatchedFee	decimal(18,2), not null		
ModifiedDate	date, not null		

ProvidedCare			
ProvidedCareID	int, not null, identity(1,1)	PK	
ProvidedCareName	nvarchar(50)		
UnitsDispatchedFee	decimal(18,2), not null		
ModifiedDate	date, not null		

			PK of
Cities			
CityID	int, not null	PK	
CityName	nvarchar(50), not null		
StateID	int, not null	FK	**States**
Location	geography, null		
ModifiedDate	date, not null		

States			
StateID	int, not null	PK	
StateName	nvarchar(50), not null		
Border	geography, null		
ModifiedDate	date, not null		

ProvidedCare			
ProvidedCareID	int, not null, identity(1,1)	PK	
ProvidedCareName	nvarchar(50)		
UnitsDispatchedFee	decimal(18,2), not null		
ModifiedDate	date, not null		

InsuranceCompanies			
InsuranceCompanyID	int, not null, identity(1,1)	PK	
InsuranceCompanyNa	nvarchar(60), not null		
ModifiedDate	date, not null		

InsuranceDeductable			
InsuranceID	int, not null	PK	
InsuranceCompanyII	int, not null	FK	**InsuranceCompanies**
InsuranceType	nvarchar(60), not null		
InsuranceDeductabl	decimal(18,2), not null		
ModifiedDate	date, not null		

MedicalNature			
MedicalNatureID	int, not null, identity(1,1)	PK	
MedicalNatureType	nvarchar(60), not null		
ModifiedDate	date, not null		