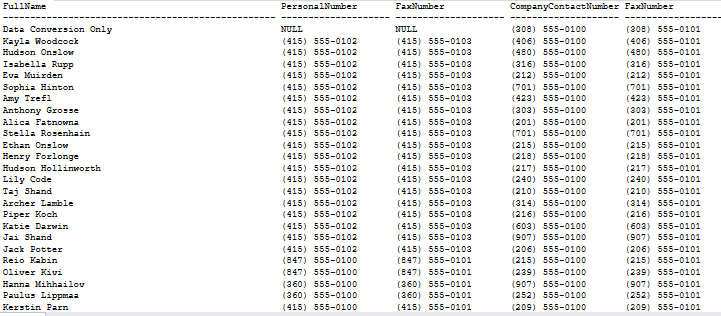
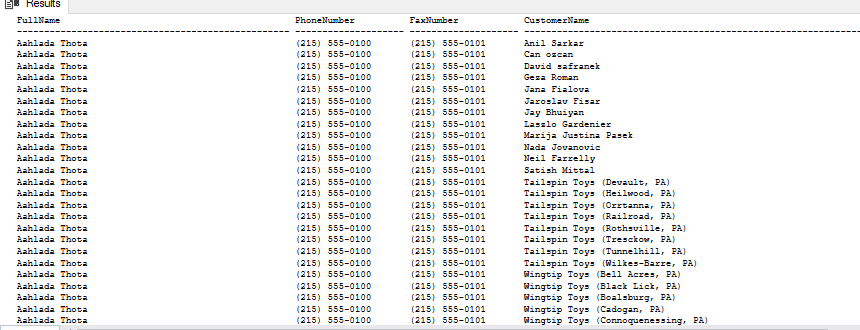
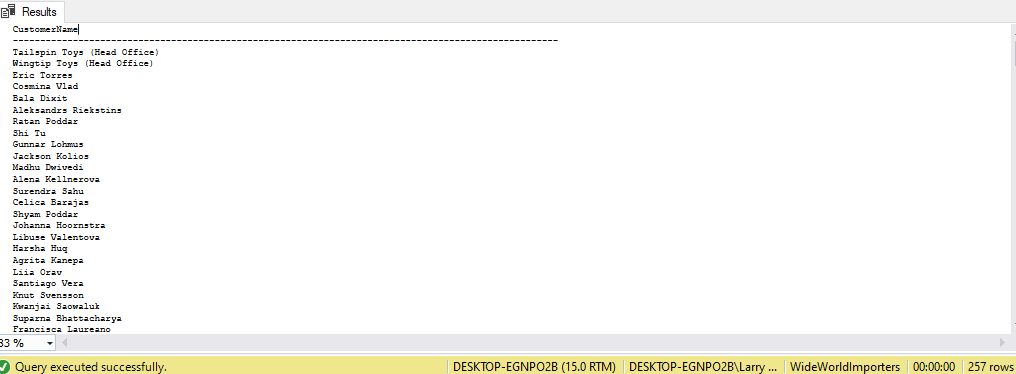
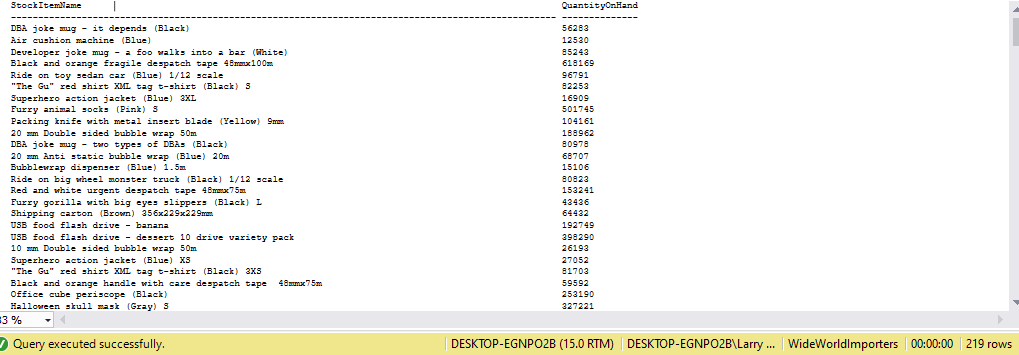
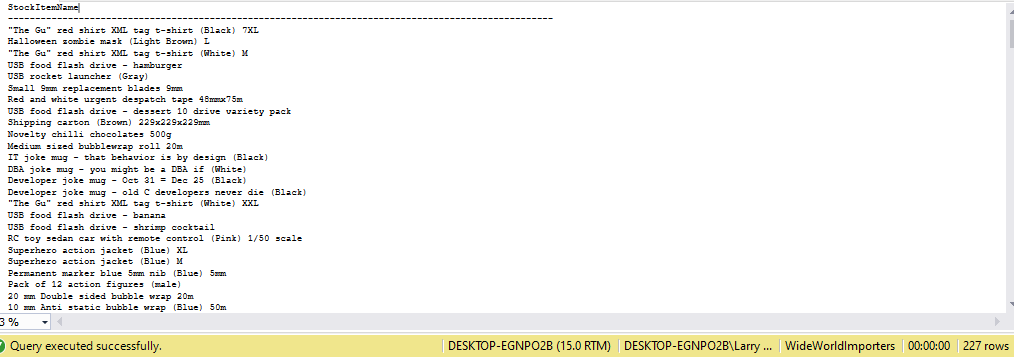
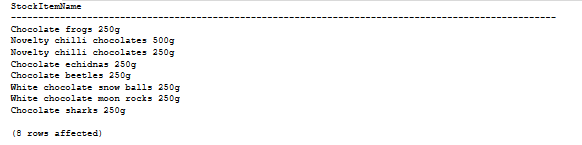
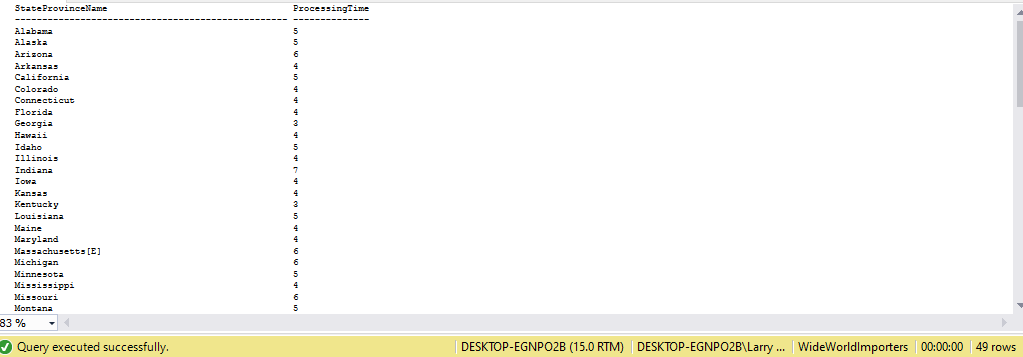
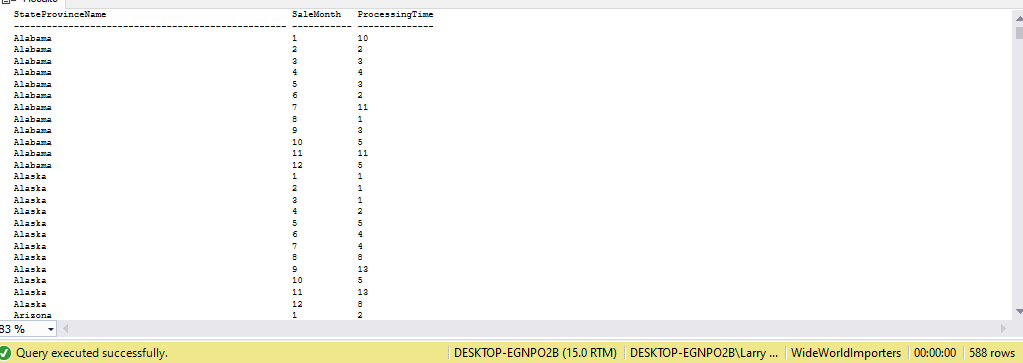
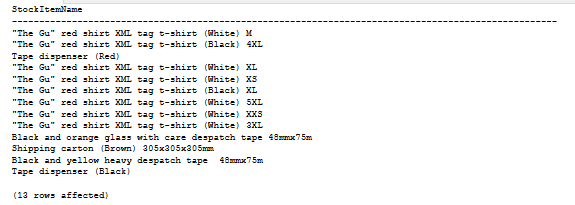
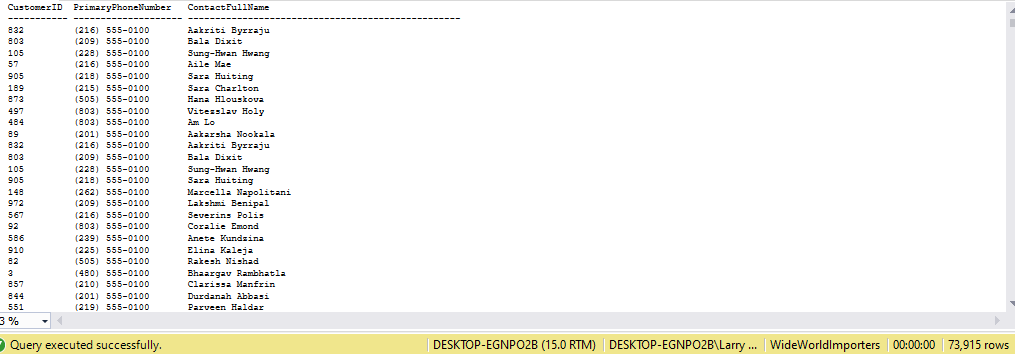
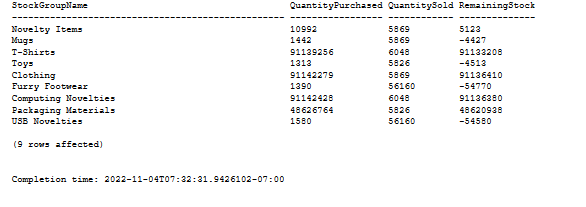
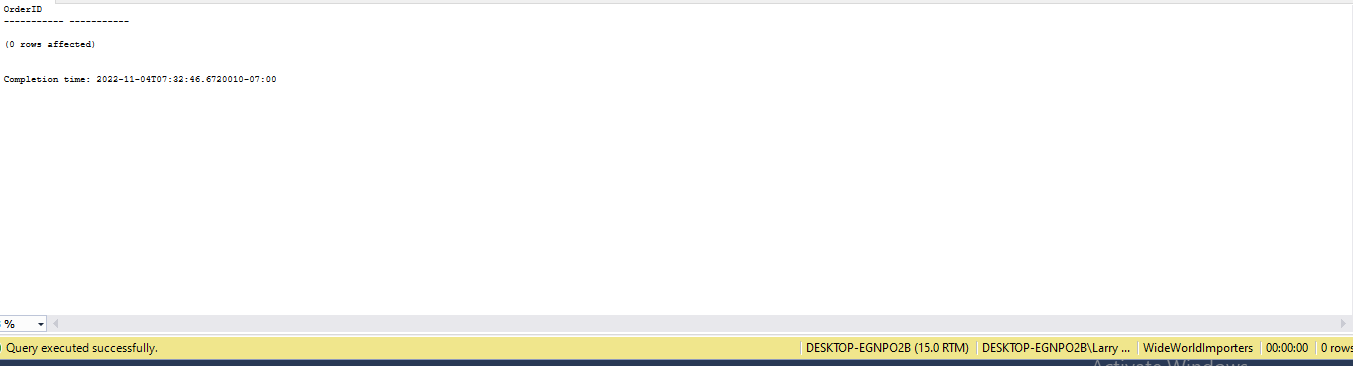
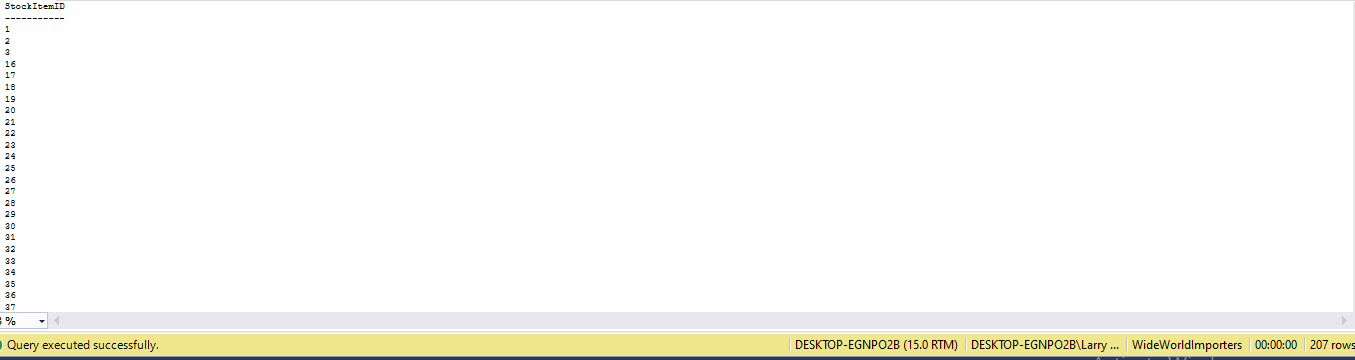
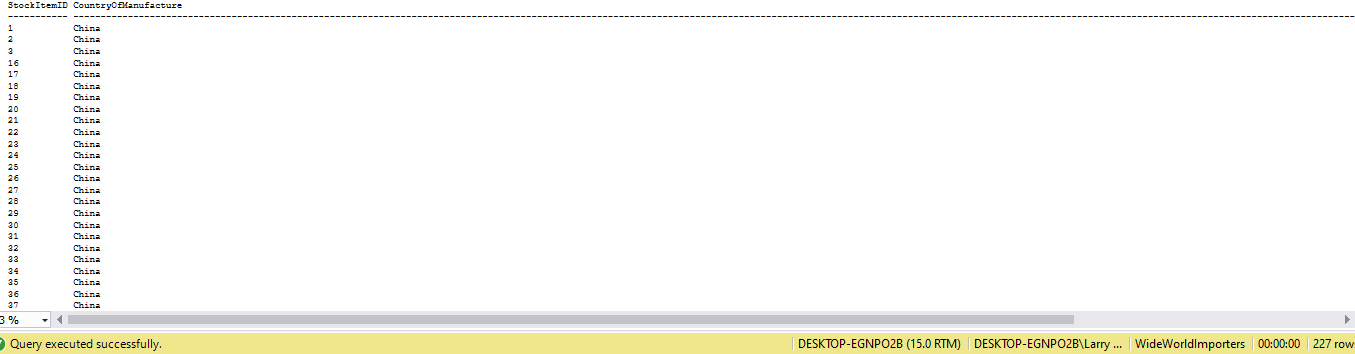
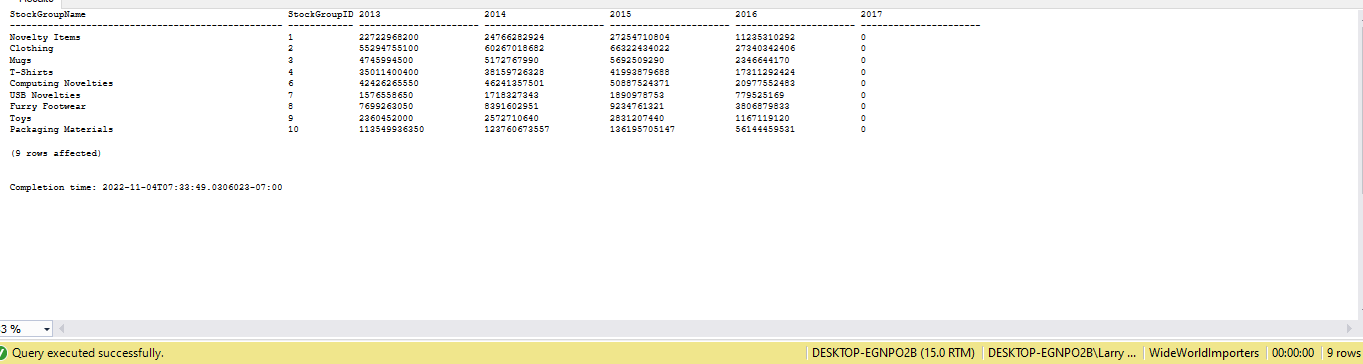
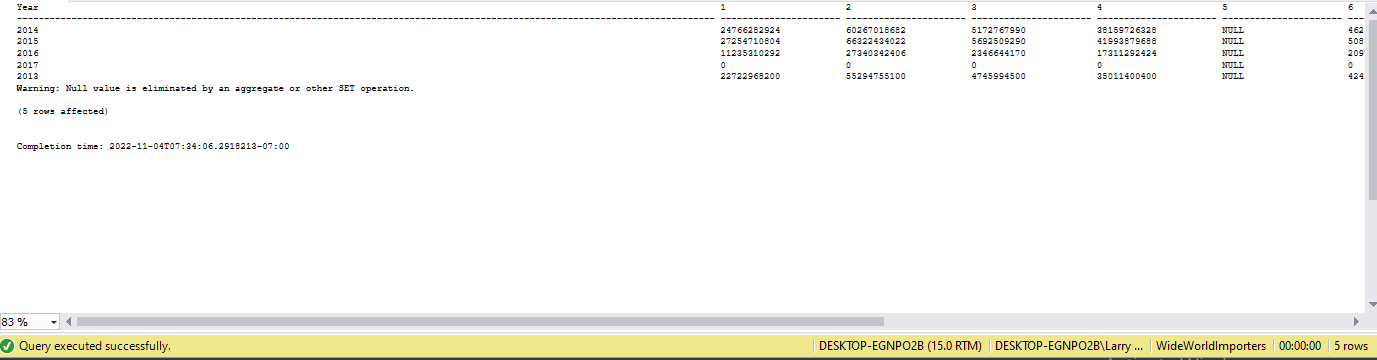
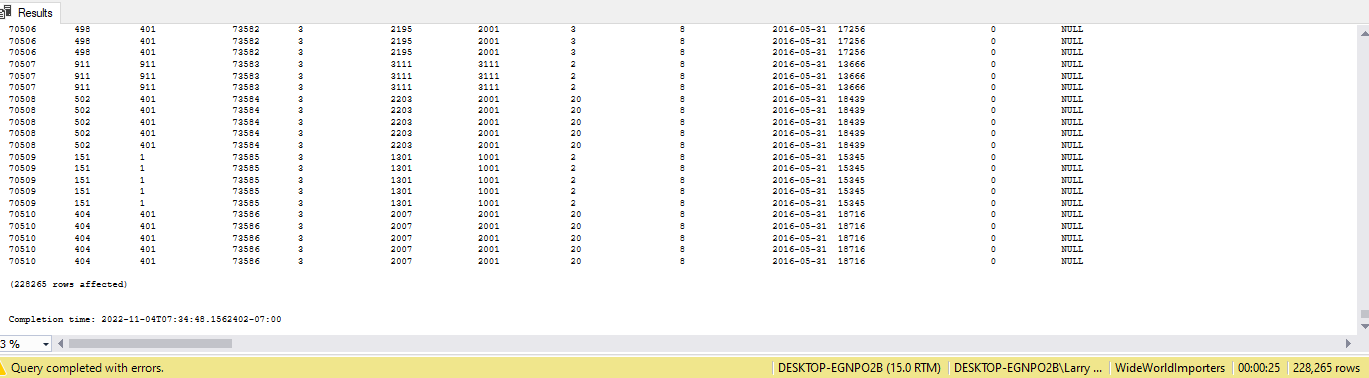
SQL Assignments

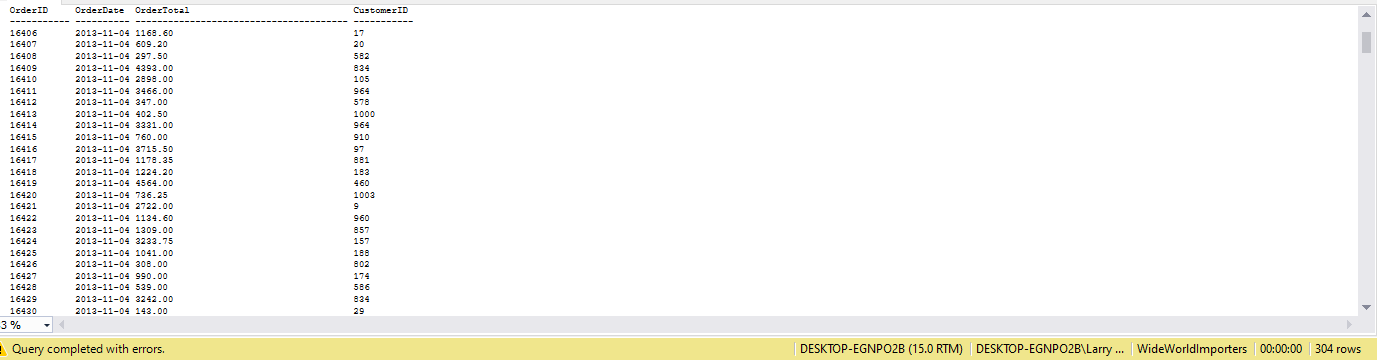
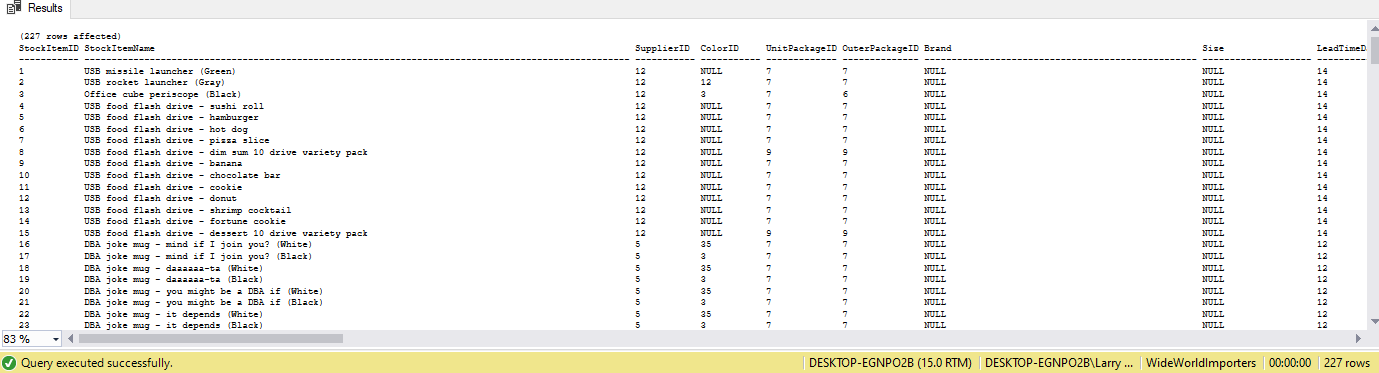
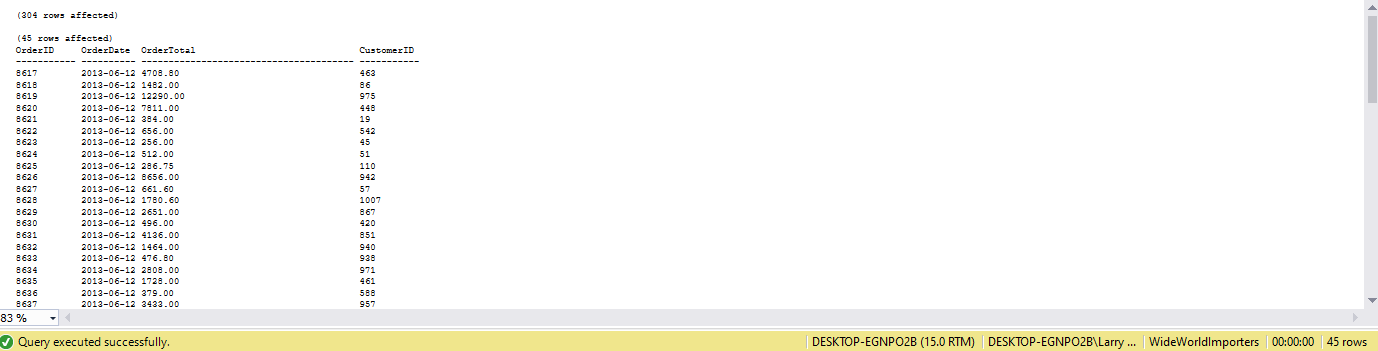
SQL related assignments will be on the Wide World Importers Database unless otherwise mentioned.

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). 
2. If the customer's primary contact person has the same phone number as the customer’s phone number, list the customer companies. 
3. List of customers to whom we made a sale prior to 2016 but no sale since 2016-01-01.
4. List of Stock Items and total quantity for each stock item in Purchase Orders in Year 2013.
5. List of stock items that have at least 10 characters in description.
6. List of stock items that are not sold to the state of Alabama and Georgia in 2014.
7. List of States and Avg dates for processing (confirmed delivery date – order date).
8. List of States and Avg dates for processing (confirmed delivery date – order date) by month.
9. List of StockItems that the company purchased more than sold in the year of 2015.
10. 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.
11. List all the cities that were updated after 2015-01-01.
12. 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.

(partially truncated - see uploaded images for full view)

1. List of stock item groups and total quantity purchased, total quantity sold, and the remaining stock quantity (quantity purchased – quantity sold)
2. 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”.
3. List any orders that had more than one delivery attempt (located in invoice table).
4. List all stock items that are manufactured in China. (Country of Manufacture)
5. Total quantity of stock items sold in 2015, group by country of manufacturing.
6. 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]
7. 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] 
8. 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.



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. 
2. 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.
3. 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.
4. 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",

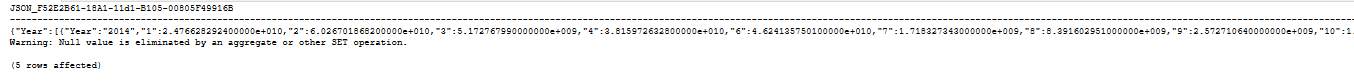
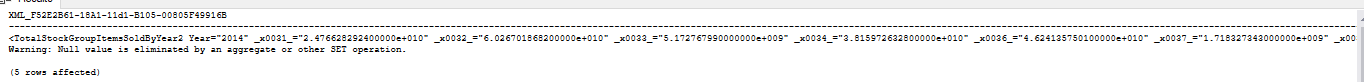
"SupplierReference":"269622390"

}

]

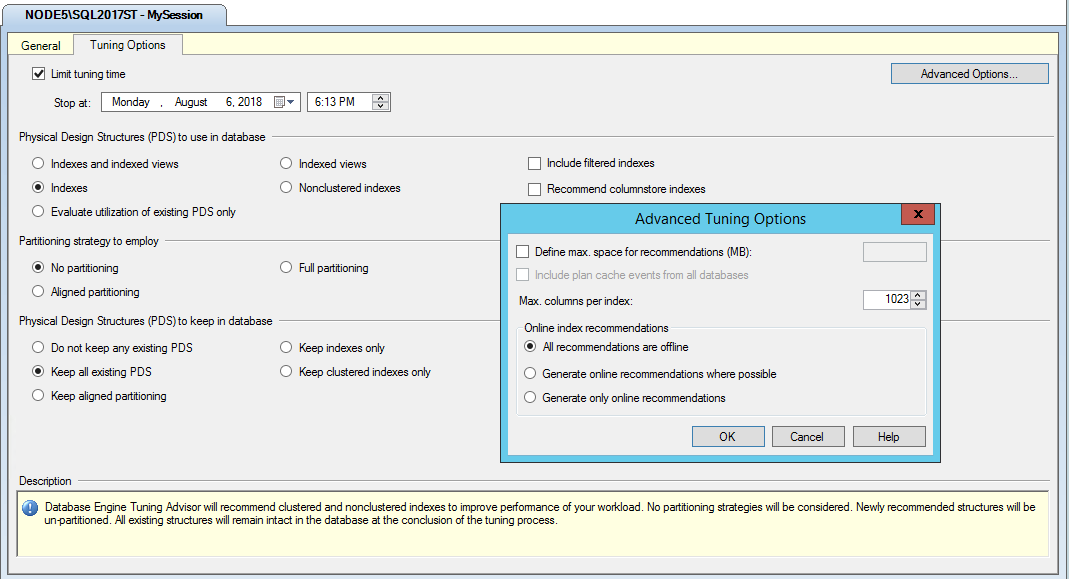
}

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.

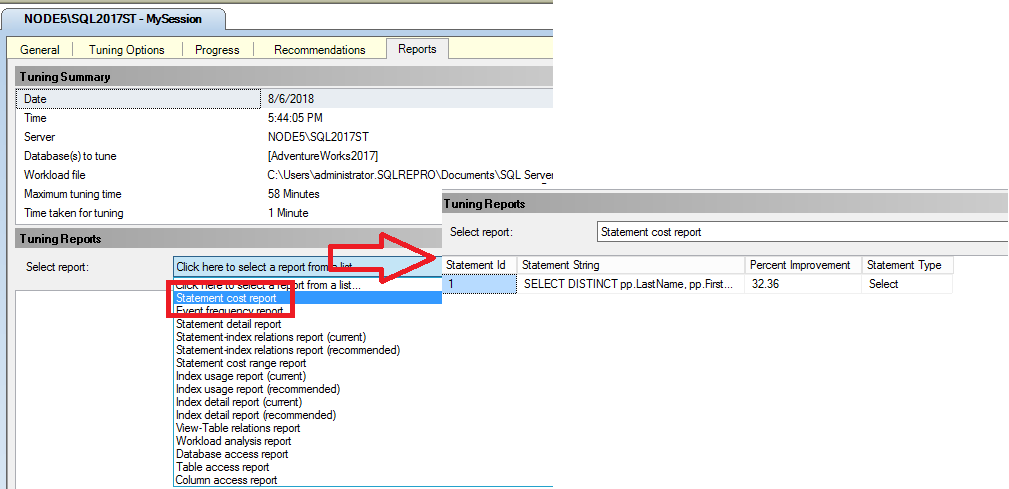
1. Revisit your answer in (19). Convert the result in JSON string and save it to the server using TSQL FOR JSON PATH.
2. Revisit your answer in (19). Convert the result into an XML string and save it to the server using TSQL FOR XML PATH.
3. Create a new table called ods.ConfirmedDeviveryJson with 3 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.
4. Write a short essay talking about your understanding of transactions, locks and isolation levels.

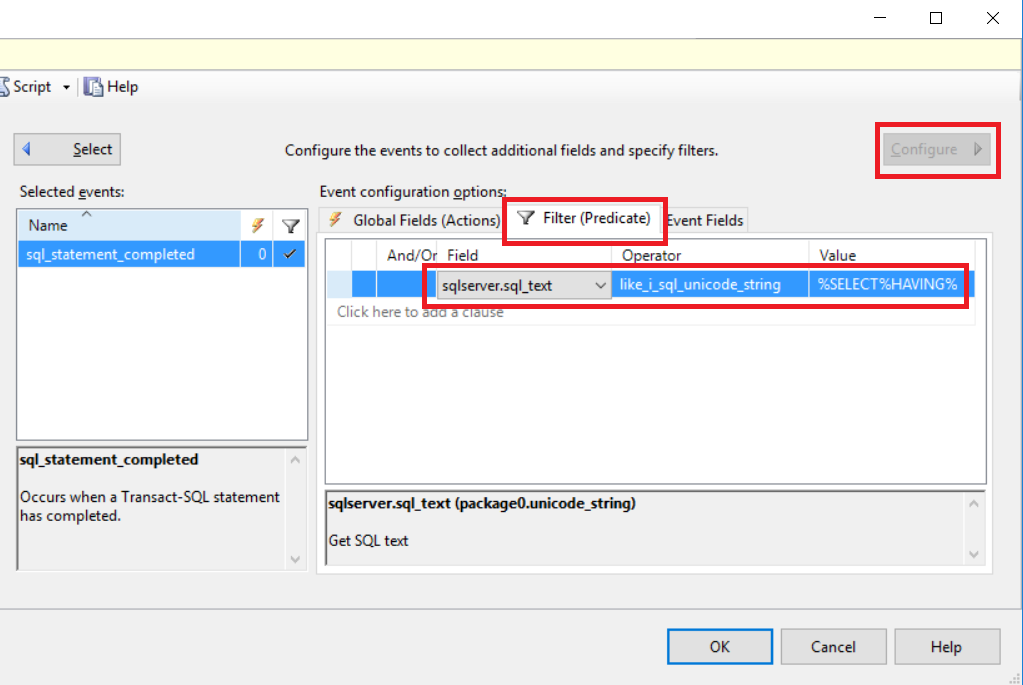
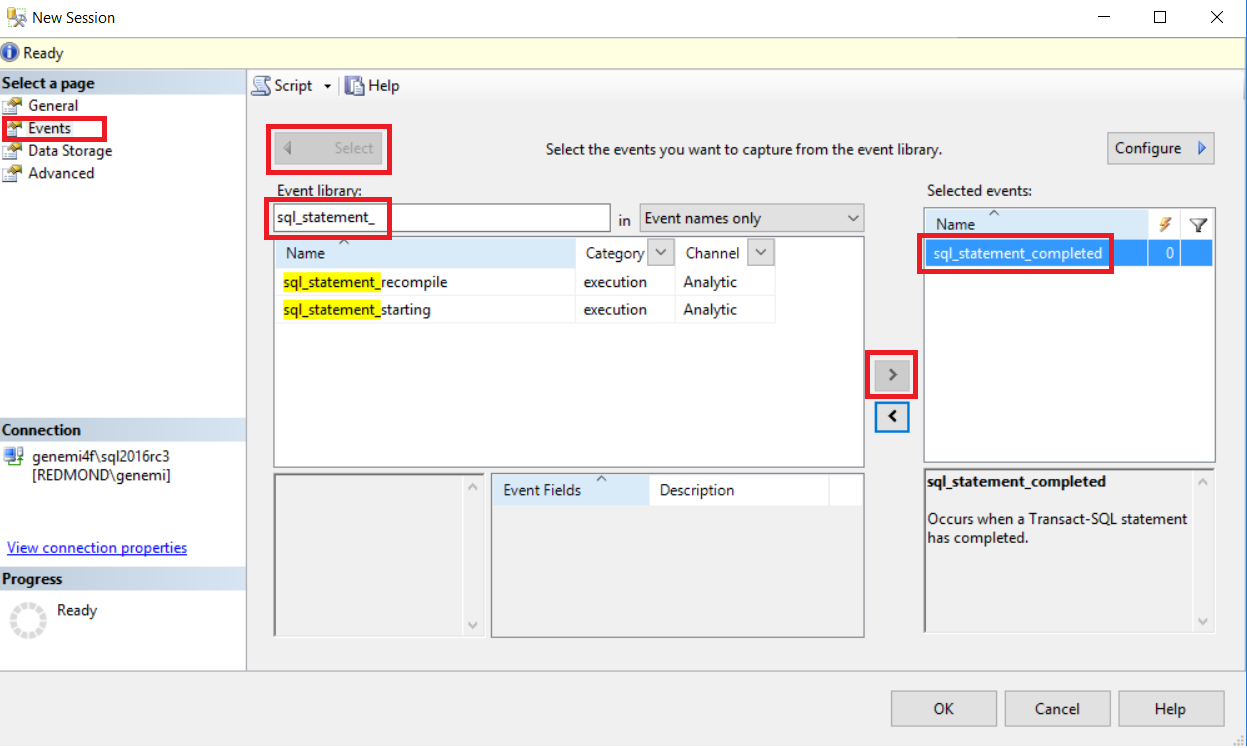
Each transaction in a SQL DB must adhere to ACID (atomicity, consistency, isolation, durability). Databases can be accessed by many readers/writers at the same time and these transactions must adhere to ACID principles to ensure there are no consistency issues. Locks are a common formulation to prevent concurrent read/writes behaving unpredictably in many systems - in the context of SQL Server engine, locks allows only one transaction to access specific objects in specific ways affected by a transaction; before the lock is released, no other transactions can access these same objects in specific ways and must request that lock and wait in the meantime. Locks in SQL Server have different modes such as exclusive, shared, and update. Transactions have an isolation level (read uncommitted, read committed, repeatable read, serializable, snapshot (note that any transaction can only be one of these states at a time)) which define how isolated one transaction is from a resource and how long a transaction can hold a lock. The choice of isolation level is often made to balance between runtime and data integrity. Shared locks for read committed or repeatable reads are generally row locks.

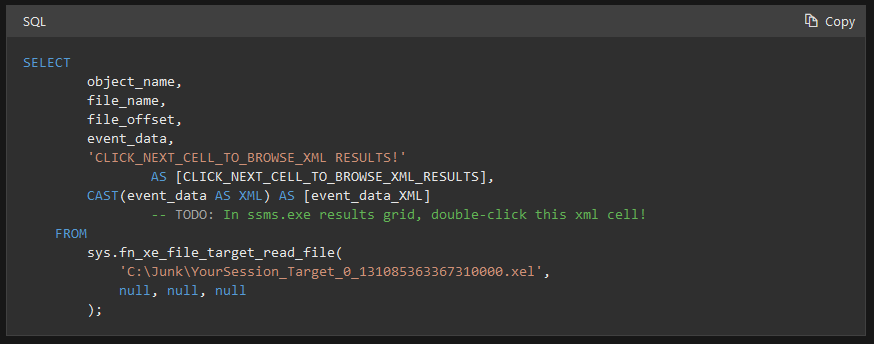
1. Write a short essay, plus screenshots talking about performance tuning in SQL Server. Must include Tuning Advisor, Extended Events, DMV, Logs and Execution Plan.

The Tuning Advisor assists a user in tuning databases for better performance for queries; often this is done through suggestions on modifying DB structures such as indexes, indexed views, or table partitions. In addition to a standard GUI, the Advisor features a **dta** utility which can be used in scripts for automated tuning. Users can configure workload files, tables, or XML files as inputs when tuning and can also interface with the GUI to select specific kinds of DB structures to tune on, as seen in the picture below.

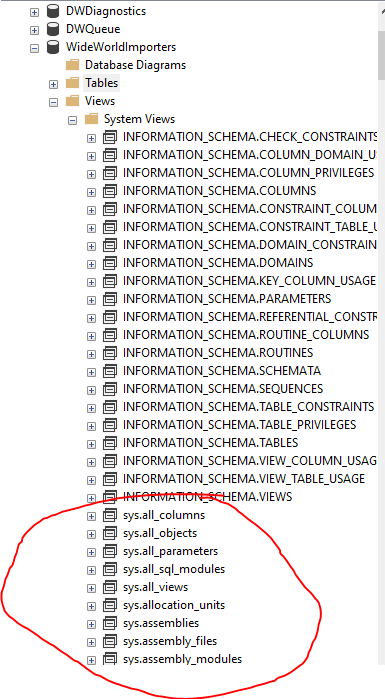
Reports show project improvements based on suggested tuning - many different kinds of reports can be exported, usually formatted as XML. All of this can be seen in the below photo:

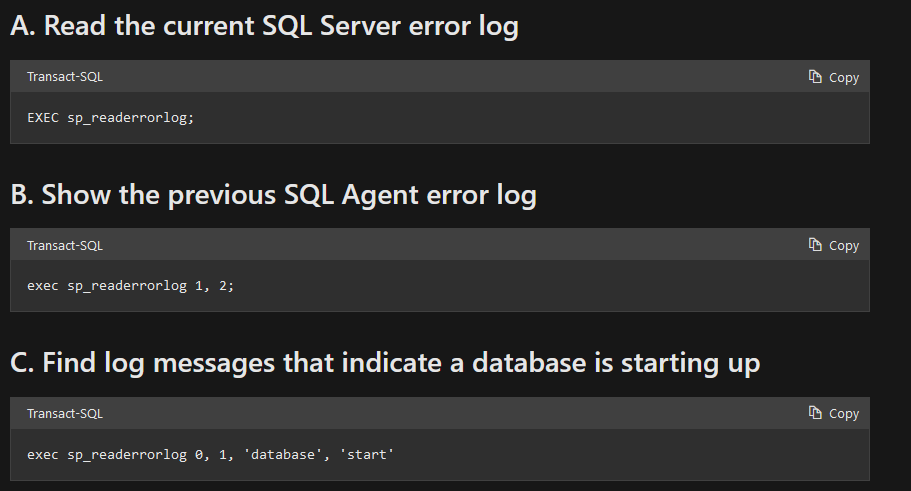


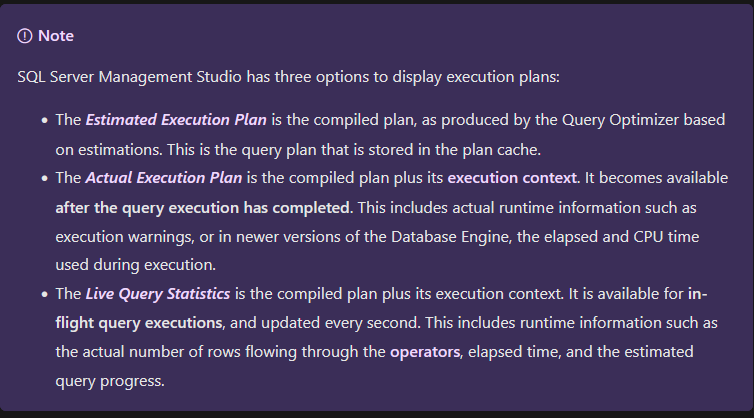
Extended events help users collect data to help monitor and troubleshoot problems within SQL Server, such events can be configured to focus on specific kinds of occurrences and displayed/saved in a variety of different ways for different use cases.

The above screenshots depicts the process of selecting a specific kind of event (in this case all SQL SELECT statements with a HAVING clause) to monitor. Results of monitoring through Extended Events can be viewed within SQL Server with the standard SELECT command used within SQL itself. The below screenshot depicts the syntax and construction of such a statement.

Dynamic Management Views (DMV) return information about server state that allows end user to monitor the health of a server instance, diagnose problems, and tune performance. DMV also includes tools to monitor replication and health status of other server instances. Similar to bog standard views in SQL server and utilize the same kind of DML, there are many kinds of DMV for specific kinds of optimization and performance tuning such as Memory-Optimized Table Dynamic Management Views , Object Related Dynamic Management Views, Query Notifications Related Dynamic Management Views. Notably, in addition to SELECT permissions on an object, querying a DMV requires VIEW SERVER STATE or VIEW DATABASE STATE permissions. The below picture indicates the location and shows a few kinds of DMVs.

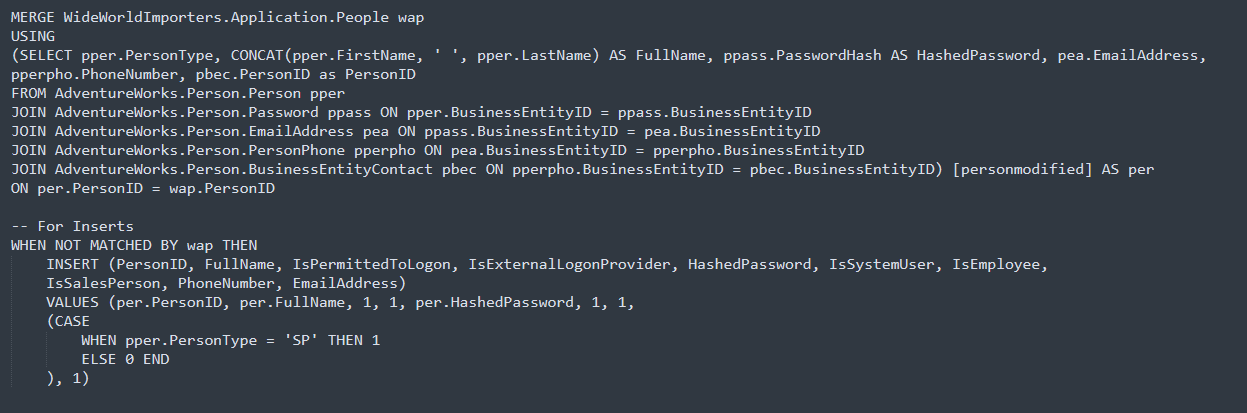


Logs are an append only time ordered sequence of records used for many different kinds of software purposes; in the most abstract conception they describe when events occur. Similar to aforementioned functionalities, the SQL Server error log contains user-defined events and also events that can help you troubleshoot; dissimilar to others in that these logs contain only errors and so help tune performance only in the sense that it can help you diagnose when errors are occurring and perhaps help you narrow down where/why they are. Like many other tools mentioned, can be queried from within SQL Server itself (examples below)

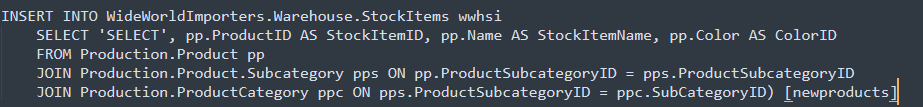
The SQL Server DB Engine uses the Query Optimizer to analyze each input query to determine the most efficient way to access data; this Query Optimizer outputs a query execution plan, which defines the order tables are accessed, how data is extracted from each table (index/table scan/index scan), and methods used to filter/aggregate/compute data.

Live statistics can provide real time info into performance and how data flows, which can be essential in debugging specific query performance issues. Access to the actual execution plan also allows you to understand resource usage, bottlenecks, and in general have a more clear picture and how and why when performance issues crop up. These plans and statistics can be accessed with DML.

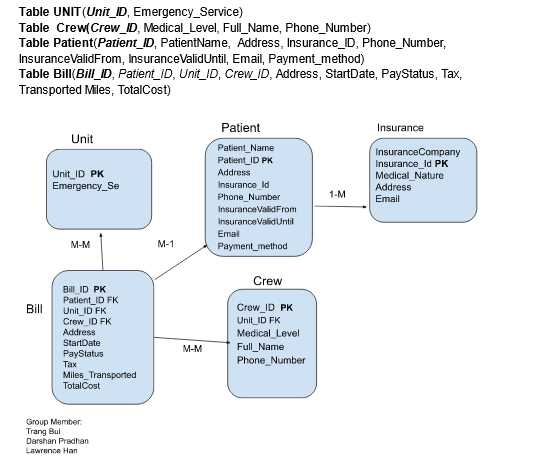
1. Write a short essay talking about a scenario: Good news everyone! We (Wide World Importers) just brought out a small company called “Adventure works”! Now that bike shop is our sub-company. The first thing of all works pending would be to merge the user logon information, person information (including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

The merge of user logon and personal information can be done with a merge statement, wherein we first create a result table based on joining relevant information from all the Adventureworks tables. Then, following the format outlined in class, we provide relevant values for each instance of insertion of a new employee record.

For the merging of products we take relevant information from the AW tables and do an INSERT



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.

~~~~

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. Luckily, we only start with 1 fact: Purchase. 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.