Module 1

Introduction to TSQL for Data Analysis

Summary

This module will present a high-level overview of contemporary organizational data architectures and workflows, and the role the TSQL language plays in these systems. It will present basic database and Microsoft SQL Server concepts including database engine instances, table relationships, and SQL as a declarative language. It will use SQL Server Management Studio to connect to an Azure SQL database and create a database diagram. Finally, it will introduce the SELECT query.

Database Queries and Analytical/Reporting Architecture

This course is about writing TSQL queries for the purpose of *making meaning from data and communicating this meaning to others*. TSQL is Microsoft's version of ANSI Standard SQL, and is the language we use to retrieve data from Microsoft SQL Server relational databases and other Microsoft data products.

So just what the heck IS a relational database, anyway? In a nutshell, its primary purpose is to act as a data management system that allows lots of users to modify the same data set at the same time without screwing up the data. Business applications use relational databases as their primary data store for this reason. Customer relationship management systems, order processing systems, student records systems, financial systems, medical records systems, etc. use the TSQL language to store and retrieve data in Microsoft SQL Server relational databases.

For our purposes, for the most part we're not going to worry about the parts of TSQL that are used to write and maintain data in the database. Our focus will be on getting data out of the database for the purpose of analyzing it, making meaning from it, and reporting this meaning to others.

SQL in the On Premises Analytical Enterprise

The diagram below shows a typical on-premises business intelligence or data analytics architecture. A business application (i.e. an order processing system) uses an MS SQL Server database as its backend. This database is designed to support efficient data modification commands rather than efficient data retrieval commands, as the critical factor for this type of application is the support of transactional processing. As business application users (i.e. store clerks, etc.) use the application, the application reads and writes data to the database using TSQL commands.

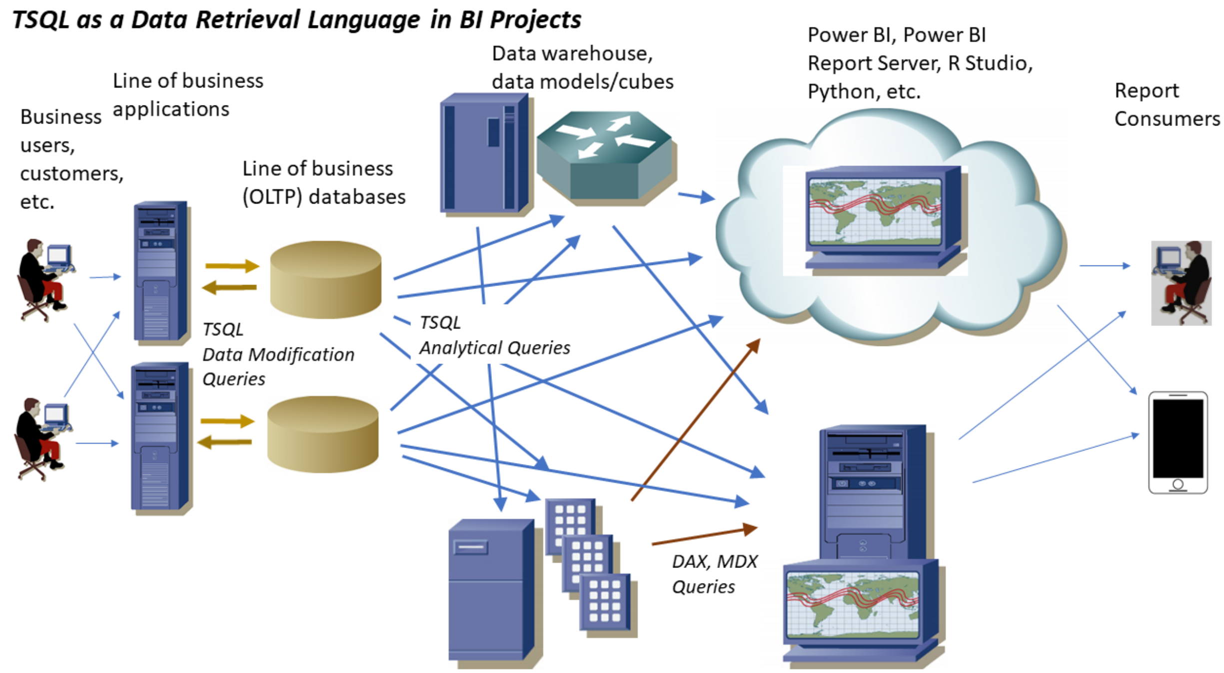


Figure 1: TSQL as a data retrieval language in BI projects

Often organizations will create a separate SQL Server database for the purpose of reporting. This offloads the overhead of report processing to another server and allows the reporting database (or *data warehouse*) to be designed for analytical SELECT queries rather than data modification commands. A collection of data copy operations (generally referred to as Extract, Transform, and Load, or ETL, routines) will run on an automated basis, usually at off hours, and keep the data warehouse updated with data from the application database.

Client tools such as Power BI can issue data retrieval commands against any SQL Server database, whether it's designed for OLTP processing or as a data warehouse. If your organization maintains a data warehouse, then this is most likely to be the source of your report data. However, reports can issue queries against an application database, too.

The Azure SQL database

In this class we'll learn to use TSQL to retrieve and analyze data from an **Azure SQL database**. An Azure SQL database is an OLTP database essentially similar to an on-premises instance of SQL Server, except that it is hosted in the Azure cloud. Everything you'll learn about TSQL will apply equally to both Azure SQL databases and on-premises SQL Server databases.

SQL in the Azure Synapse Analytics Environment

Many organizations have moved to a cloud-based environment for analytical services. such as Azure Synapse Analytics. The Azure cloud platform offers pay-for-use services including relatively unlimited computing resources within Azure Synapse Analytics' single, cohesive platform.

In the diagram below, Azure Synapse Analytics becomes the primary analytical platform. Here, TSQL can be used to extract, transform, and load data into data stores within Azure Synapse Analytics, to analyze data within Azure Analytics, and to load data from Azure Synapse Analytics into client tools for further analysis and use.

In this class we'll learn to use TSQL to retrieve data from text files we've uploaded to Azure Data Lake Storage. As we'll see, often we'll need to analyze data in large, semi-structured text files such as machine logs and social media feeds. We'll learn to use Azure Synapse Analytics serverless SQL pool to extract and analyze data from these types of files.

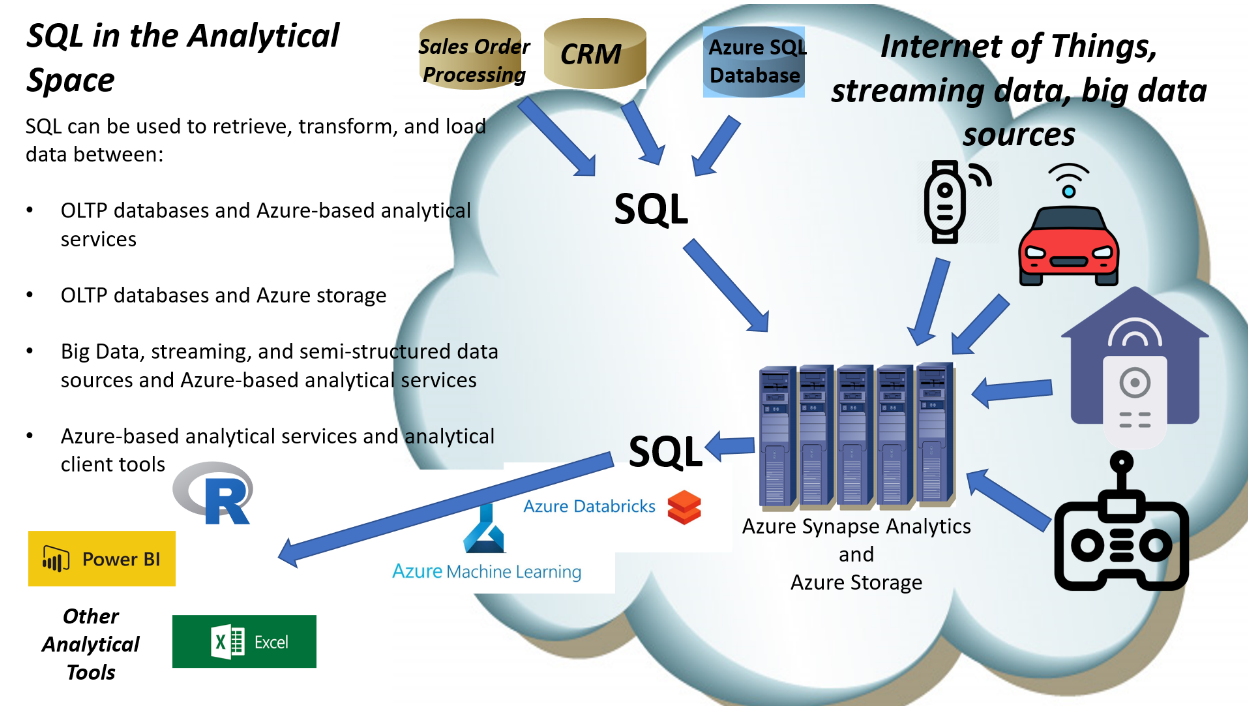


Figure 2: SQL in the contemporary analytical space

Basic Database Concepts / Prerequisite Knowledge

Before we get too deep into the SELECT statement, let's make sure we're clear on a few basic terms and concepts.

1. ***SELECT statement***. The SELECT statement is a specific command within the TSQL programming language used to return data from a database server. This will be the primary focus of this course.
2. ***Database Engine***. The database engine is the piece of server software running on a server computer or within a cloud environment that manages databases and processes TSQL language statements. Throughout most of this class we will use the Azure SQL database engine.
3. ***Database server*** or ***Server instance***. This can be a little tricky to understand. A database engine can be installed multiple times on the same physical (or virtual) server machine. Each time it is installed it acts as a completely separate SQL server. One machine, for example, may have four

copies of the engine installed on it, and each copy acts as a completely separate SQL server. The term ***Database server*** or ***Server instance*** refers to a single one of these engines. Instances do not share databases or security, and are fully independent of one another.

4. ***Default instance.*** The copy of the database engine installed on a given server machine configured to listen to default SQL port 1433 is called the Default Instance.
5. ***Named instance.*** All instances other than the default instance are called Named Instances. Each named instance listens to its own TCP port number.
6. ***Database***. A database is a specific collection of tables, views, stored procedures, and other data-related objects. A server instance can contain multiple databases. Databases are completely contained within server instances. In other words, you couldn't have part of a database on one instance and part on another, and instances cannot share databases.
7. ***Table.*** A table is the primary object within the database that contains data. Tables are composed of rows and columns.
8. ***Record.*** A record is a single row in a table. Can also be a single row in a result set produced by a SELECT query.
9. ***Recordset.*** A recordset is a collection of records. Usually we will refer to the collection of records produced by a SELECT query as a recordset or result set or simply dataset.
10. ***Azure SQL Database.*** An Azure SQL database is a SQL database running on an Azure cloud-based server instance rather than an on-premises server instance.

Query Development Tools

SQL Server Management Studio (**SSMS**) is the primary tool we're going to use to work with Microsoft SQL Server databases. SSMS is the most commonly used tool for creating databases, creating database server logins and applying security to databases, for creating tables and other objects within databases, and for writing TSQL queries.

In this course our primary focus will be on writing SELECT queries using SSMS. The SELECT query is the TSQL statement used to retrieve data and create datasets from MS SQL Server databases.

Creating the CarDealer Database

One of the benefits of cloud-based services such as the Azure SQL database is that there is no up-front cost to deploying the service. Rather, you pay for Azure services in a metered way that allows you to only pay for what you use. As we'll see, the cost of a small Azure SQL database capable of only a few transactions per second can be as low as $5/month. However, this service can be immediately allocated additional resources (at an additional cost) making it capable of processing many thousands of transactions per second.

In the following walk-through, we're going to install an Azure SQL database named CarDealer. This is the database we'll use throughout most of the course.

Prior to beginning the walkthrough, you're going to need:

1. A Microsoft account with an Azure subscription;
2. The course files extracted into a local folder. This walkthrough will assume the local file path of your course files is C:\SQLForDA\.
3. SQL Server Management Studio (SSMS) installed on your local machine.

Walkthrough: Creating the Azure SQL Database Logical Server

1. Log into the **Azure portal** at **https://portal.azure.com**.
2. Click the “hamburger” icon (the three –‘s in the upper left corner) and select ***SQL databases > Create***.
3. On the **Basics** blade of the **Create SQL Database** dialog, select your **Subscription**.
4. Under **Resource Group**, click ***Create new***, and create a resource group named **CarDealer**.
5. Enter **AdventureworksLT** for the database name.
6. Beside **Server**, click ***Create new***, and in the **Create SQL Database Server** dialog, use the following:
    a. Server Name: Use any unique server name, such as ***petunia531*** or ***hotdog420***.
    b. Choose the major Azure data center closest to your location, such as **East US** or **West Europe**.
    c. Under **Authentication method** select **Use SQL Authentication**.
    d. For the **Server admin login** use **Student**, and for the **Password** use **Pa55w.rd**.
    e. Click **OK**.
7. Select **Development** beside **Workload environment**.
8. Beside **Compute + storage**, click **Configure database**.
    a. In the **Configure** dialog, in the **Service tier** drop-down list, select **Basic** under **DTU-based purchasing model**.
    b. Click **Apply**.
9. Click **Next: Networking >.**
10. Beside **Connectivity method** select **Public endpoint.**
11. Beside **Allow Azure services and resources to access this server** choose **Yes.**
12. Beside **Add current client IP address** choose **Yes**.
13. Click **Next: Security >**.
14. Click **Next: Additional settings >**.
15. Beside **Use existing data**, select **Sample**. If the AdventureworksLT pop-up appears indicating a change will be made to the Compute + Storage settings, select **OK**.
16. Click **Review + create**, then click **Create** to submit the Azure SQL database for deployment.
17. When deployment has completed for the *AdventureworksLT* database above, navigate to your list of SQL databases by clicking **SQL databases** under the hamburger icon.
18. Click the *AdventureworksLT* database, and find the **Server name** on the **Overview** page. This is the name of your **logical SQL server**. Copy it to the clipboard.

Walkthrough: Importing the CarDealer Database

At this point you've got a logical SQL Server created, with one database (AdventureworksLT) attached to it. AdventureworksLT is a sample database Microsoft makes available as a learning tool. We won't be using it in this course. However, once the course is over, it is highly recommended that you use AdventureworksLT to further practice the SQL skills learned in this class.

We're going to be using a different database designed specifically for this class. It's named **CarDealer**, and the steps below walk through the process of importing CarDealer to your logical Azure SQL server from a local .bacpac file.

19. Open **SQL Server Management Studio** on your local desktop. When the SQL Server dialog appears:
    a. Ensure **Database Engine** is selected as the **Server Type**.
    b. Enter (or paste) the name of your logical SQL server in the **Server name** box;
    c. Select **SQL Server Authentication** in the **Authentication** drop-down list
    d. Enter *Student* for **Login** and *Pa55w.rd* for **Password**.
    e. Click **Connect**.
20. In the **Object Explorer** window, click the small plus sign (**+**) to left of the server name to expand the hierarchy and expose the ***Databases***, ***Security***, and ***Integration Services Catalogs*** folders.
21. Right-click **Databases**, and select **Import Data-tier Application**.
22. Click **Next >** on the **Introduction** page.
23. On the **Import Settings** page, click **Browse**, then select ***C:\SQLForDA\CarDealer.bacpac*** and click **Open**. Click **Next >**.
24. On the **Database Settings** page, enter the following values and click **Next >**:
    a. New database name: **CarDealer**.
    b. Edition of Microsoft Azure SQL Database: **Basic**
    c. Maximum database size (GB): **2**
    d. Service Objective: **Basic**
25. On the **Summary** page click **Finish**. A new database named ***CarDealer*** will be added to your Azure SQL database logical server. It can sometimes take several minutes to complete. When **Operation Complete** appears, click **Close**.
26. In the **Object Explorer** window within SQL Server Management Studio, right-click the **Databases** folder and select **Refresh**. You should see the two databases you installed:
    a. **AdventureworksLT** – you installed this by selecting **Sample** at the **Use Existing Data** prompt during the setup of the first database within the Azure Portal.
    b. **CarDealer** – you installed this database by importing a **.bacpac** file from your local machine using SQL Server Management Studio.
27. Just to make sure everything installed correctly, expand the **CarDealer** database, expand Tables, right-click the **dbo.Customers** table and choose **Select Top 1000 Rows**. A query window should open and a SELECT query should automatically run against the **CarDealer** database, returning all the records from the **dbo.Customers** table. Close the query window but leave SSMS open.

Tables and Relationships in the CarDealer Database

Let's explore the **CarDealer** database in SSMS.

On the left-hand side of SSMS you'll see the **Object Explorer** window. This is a hierarchical list of all of the objects on this instance (in this case, the logical server). Click the plus sign to the left of the **Databases** folder to expand it. Under the Databases folder is a list of all user-created databases, in addition to a subfolder containing the system databases.

Before we can write any queries, we need to understand the tables and relationships in the CarDealer database. One of the quickest and easiest ways to do this is to create a database diagram by right-clicking on the **Database Diagrams** folder under the CarDealer database in Object Explorer and choose ***New Database Diagram***. You can then add all of the tables to the diagram.

Before we can create a diagram for a database, however, we need to install a collection of support objects by clicking **Yes** when the message prompt shown in Figure 3 appears:

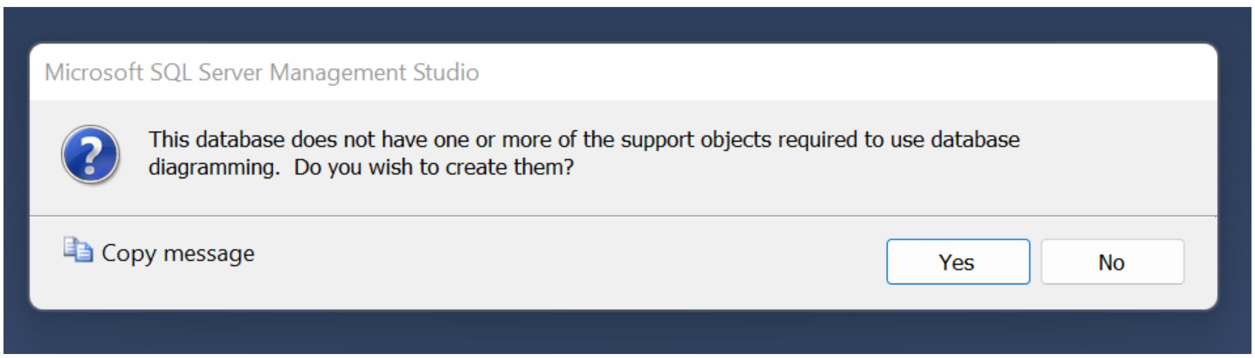


Figure 3: Click Yes to install objects required for diagrams

Once the tables have been added to the diagram, you'll generally want to zoom out (the **Zoom** feature is in the right-click menu – right-click anyway on the diagram background to pull it up) so you can see them all. **Table relationships** are represented by the lines that connect one table to another. You'll want to re-arrange the table so that the table relationships are all clearly visible.

Figure 4 below contains a diagram of the CarDealer database:

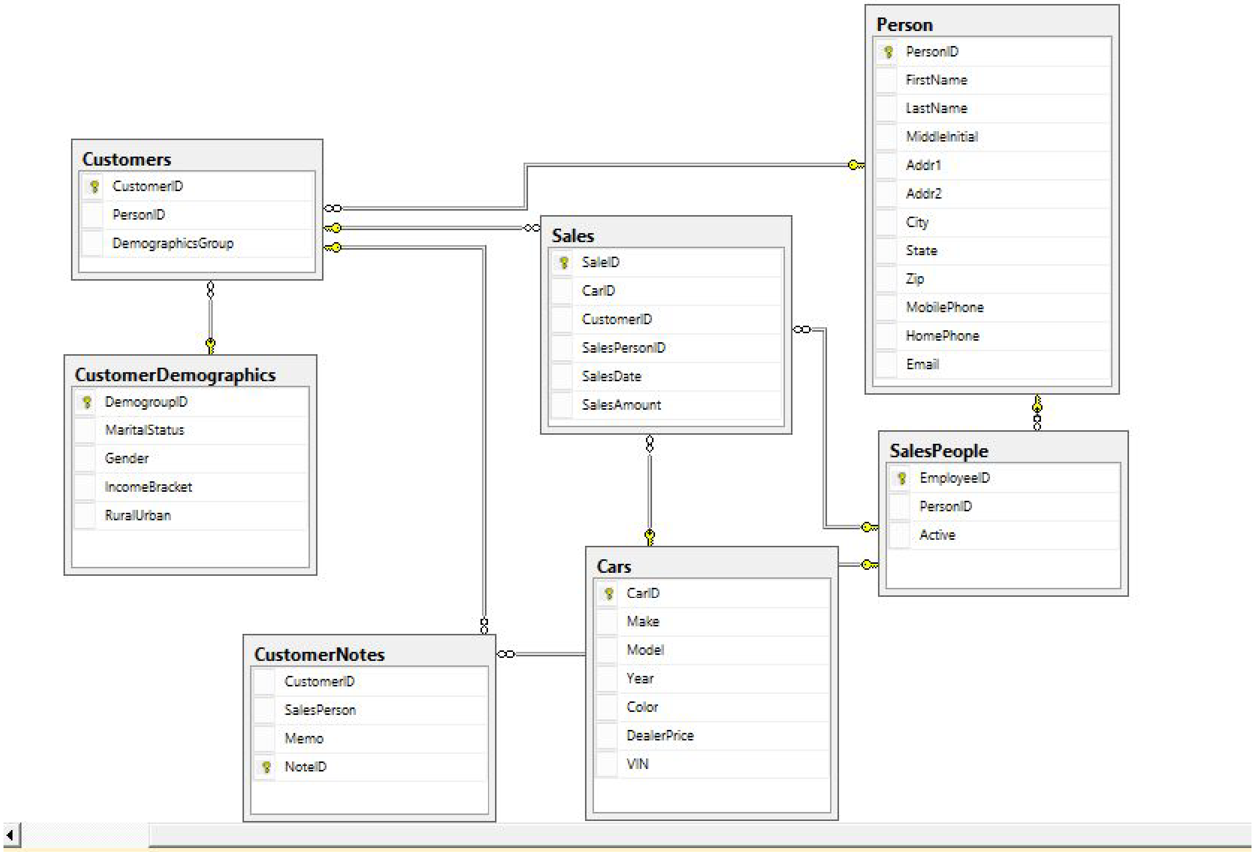


Figure 4: Diagram of the CarDealer database

The arrows in the diagram indicate the table relationships. Right-clicking an arrow and choosing **Properties** will bring up the Properties page for the relationship. Click the ellipsis button to the right of **Tables and Columns Specification** to see the specific primary key – foreign key pair in the relationship. Before you begin writing reports against any database, it's important to understand the tables, columns, and relationships.

Figure 5 shows the menu that appears when right-clicking the relationship arrow between the Sales table and Cars table. It also shows the expanded **Tables and Columns Specification**, in which the related columns can be seen. We'll introduce table relationships in much more detail in a later module. However, the idea is quite simple. In the example in Figure 5, we see that every row in the Sales table has a matching value for the CarID column in the Cars table. We know that because the CarID column in the Sales table is listed as a **Foreign Key** column, and the CarID column in the Cars table is listed as a **Primary Key** column. That's really all it is. Later in the course we'll talk about what this means for our Select queries.

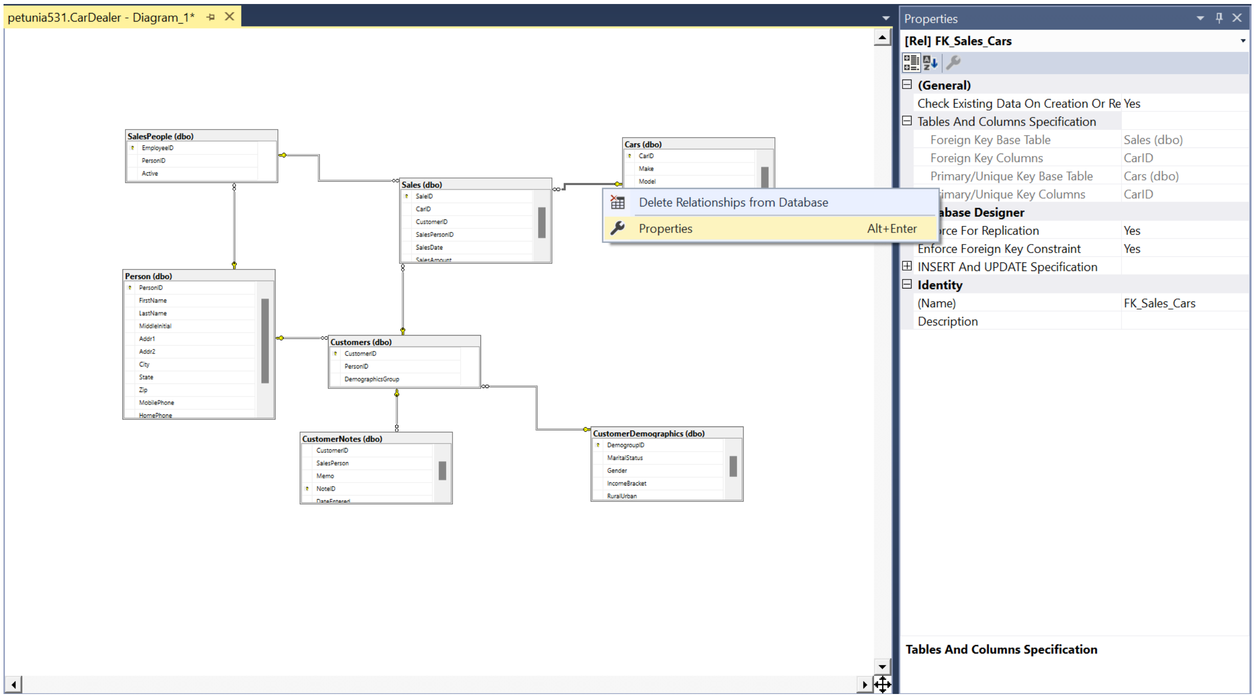


Figure 5: Digram of CarDealer database, right-click menu of relationship between Cars and Sales tables, and relationship Properties panel showing the related columns

Now that we've created a diagram, let's expand the CarDealer database in Object Explorer by clicking the plus sign to the left of it. This will display a list of folders containing all of the objects within the CarDealer database.

Now expand the **Tables** folder. Within this folder you'll find subfolders containing system tables and file tables, along with a list of the data tables in the CarDealer database. One of these tables is called **dbo.Cars**. Expand dbo.Cars, and you'll see a list of subfolders containing all of the objects within the dbo.Cars table.

Now expand the **Columns** subfolder within the dbo.Cars table, as seen in Figure 6. This will expose a list of the columns within the dbo.Cars table. Each column has three important properties:

1. A name;
2. A **data type.** The data type indicates the type of values that can be stored in the column. For example, **int** means integer, **varchar(50)** means a variable length character string up to 50 characters in length, and **money** means…. money.

3. Nullability. This simply indicateswhether or not the column will allow **null** values.

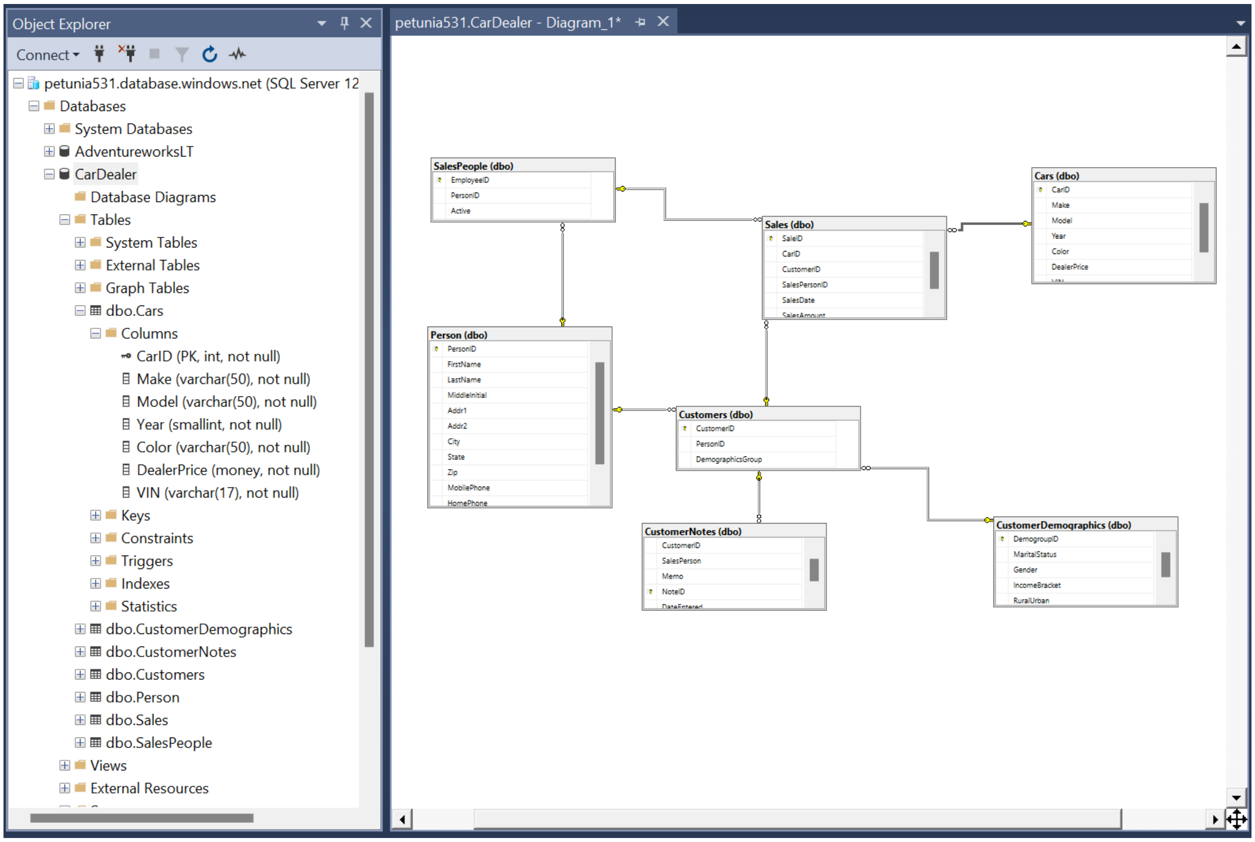


Figure 6: The Cars table in the CarDealer database expanded within Object Explorer to display the table columns and their properties

Introduction to the SELECT Query

Let's retrieve all of the data from the dbo.Cars table. To do this, we need to launch a query window. There are a couple of ways to do this. You can click the **New Query** button, or right click the CarDealer database and select **New Query**.

Since the CarDealer database was the active object in Object Explorer when we launched the new query, it will automatically appear in the **Available Databases** window at the top of the drop-down list. This database is said to have **current focus** for the query window. Any commands you run in query window will be run against the database with current focus.

The other database in the Available Databases list is **master**. The master database contains all of the system objects and meta data for the Azure SQL logical server. With the exception of a few administrative tasks, you won't be running queries against master.

The following SELECT query will return all data in the dbo.Cars table.

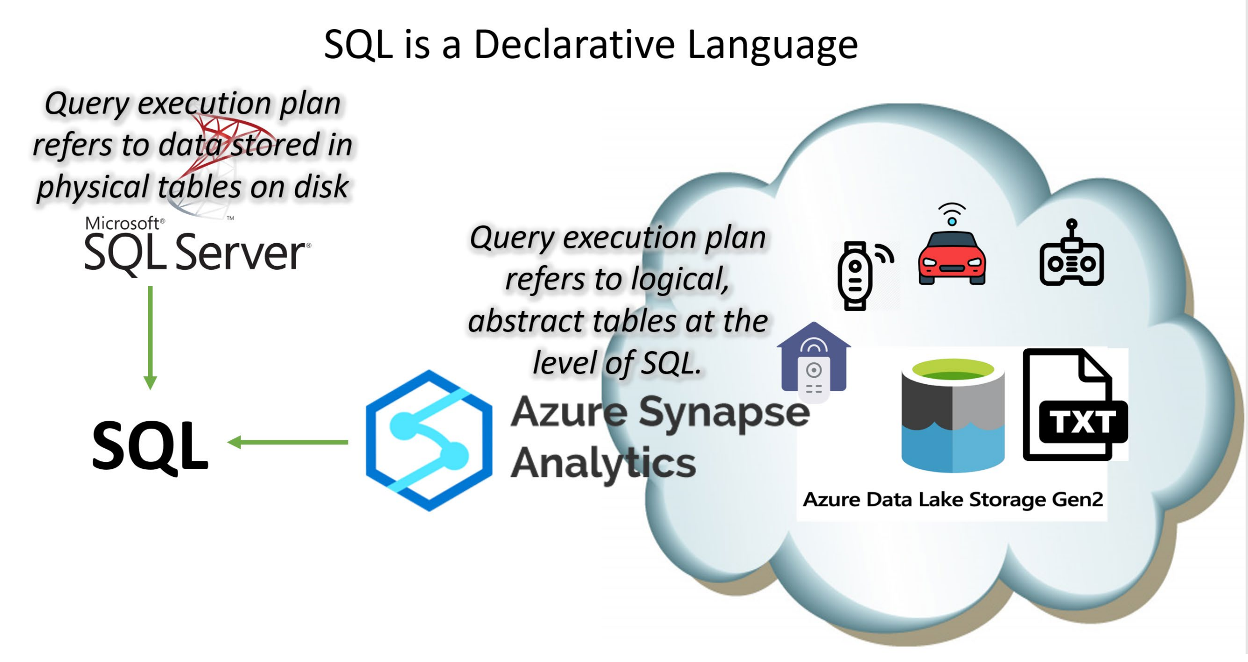
```
SELECT
	CarID
	,Make
	,Model
	,Year
	,Color
	,DealerPrice
	,VIN
FROM dbo.Cars;
```

Type that into the query window and click the **Execute** button (or just press F5). A recordset containing all of the data from dbo.Cars will appear in the Results panel below the query window.

Notice that the query ends with a semicolon. While this is technically part of the TSQL syntax, in most cases it is unenforced by the query engine.

Declarative vs. Procedural Languages

There are a couple of important differences between the TSQL language and other languages with which you may be familiar. Most computer programming languages are *procedural* in that your code provides a procedure for the computer to follow in order to produce results. TSQL is not procedural but *declarative*. You write expressions that define *sets* and operations to perform on those sets. The query processor then chooses a procedure for performing whatever set operations you've told it to perform.



The SELECT query is an example of that. You're telling the query processor that you want a set of data from the dbo.Cars table containing the contents of the CarID, Make, Model, Year, Color, DealerPrice, and VIN columns. Behind the scenes the query processor develops a procedure to follow to produce the correct result set.

The list of columns following the SELECT keyword is called the select list. You can request any of the columns, as in the following examples:

```
SELECT
       Make
       ,Model
       ,Year
FROM dbo.Cars;

SELECT
       CarID
       ,VIN
FROM dbo.Cars;
```

Remember, this is declarative rather than procedural. The FROM clause is actually processed before the SELECT list. The FROM clause specifies the table from which records are to be retrieved. The SELECT list identifies the columns to be returned from the table specified in the FROM clause.

What appears in the SELECT list are really expressions that evaluate to scalar values. In the previous examples, the column names produce a value from each row in the table. However, we could use most any expression to represent an item in the SELECT list. For example:

```
SELECT
       CarID
       ,'Great Car!'
       ,Make
       ,Model
       ,Year
FROM dbo.Cars;
```

In this case, we've used a string literal (just a string delineated by single ticks) as an expression.

In the next module we'll go into greater detail on using expressions in the SELECT list.

Additional Reading

Microsoft Azure Data Fundamentals: Explore relational data in Azure. A self-paced tutorial on basic relational database concepts. https://docs.microsoft.com/en-us/learn/paths/azure-data-fundamentals-explore-relational-data/

Transact-SQL Reference. The complete TSQL language reference. https://docs.microsoft.com/en-us/sql/t-sql/language-reference?view=sql-server-ver16

Ben-Gan, I., Machanic, A., Dejan, S., Farlee, K. (2015). *T-SQL Querying*. Published by Microsoft Press. A must-read for anyone seeking to develop advanced skills with SQL on a Microsoft platform.

Codd, E. F. (1981) *Relational Database: A Practical Foundation for Productivity*. E F Codd is generally considered the father of the relational database. This is his 1981 ACM Turing Award lecture, and a chance for us to hear from the source. https://dl.acm.org/doi/10.1145/358396.358400