# Data Definition Language SQL Data Types

.NET CORE

*A software system used to maintain a relational database is called a Relational Database Management System (RDBMS). Many relational database systems use* ***Structured Query Language (SQL)*** *for querying and maintaining a database.*

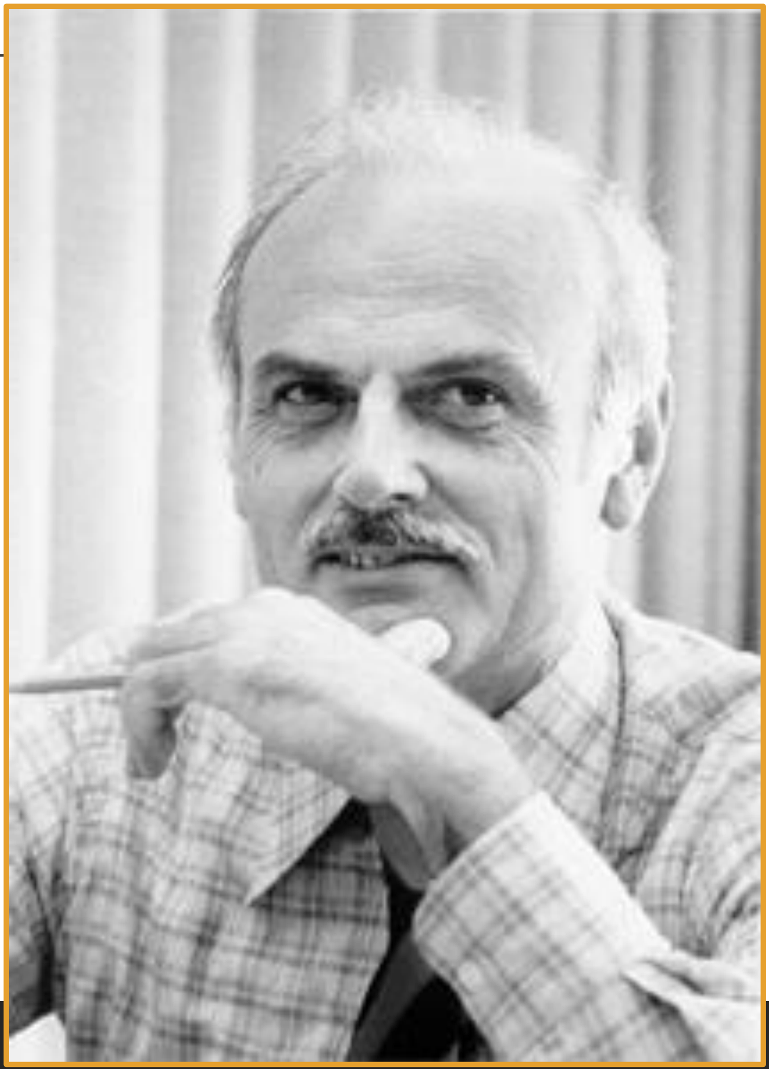
[HTTPS://DOCS.MICROSOFT.COM/EN-US/SQL/T-SQL/LANGUAGE-REFERENCE?VIEW=SQL-SERVER-VER15](HTTPS://DOCS.MICROSOFT.COM/EN-US/SQL/T-SQL/LANGUAGE-REFERENCE?VIEW=SQL-SERVER-VER15)

## (RDBMS) Relational Database Management System – History

https://en.wikipedia.org/wiki/Relational_database

Relational databases are based on the relational model of data, as proposed by E. F. Codd in 1970.

Edgar Frank "Ted" Codd (August 23, 1923 – April 18, 2003) was a British computer scientist and winner of the 1981 Turing Award.



# SQL (Structured Query Language)

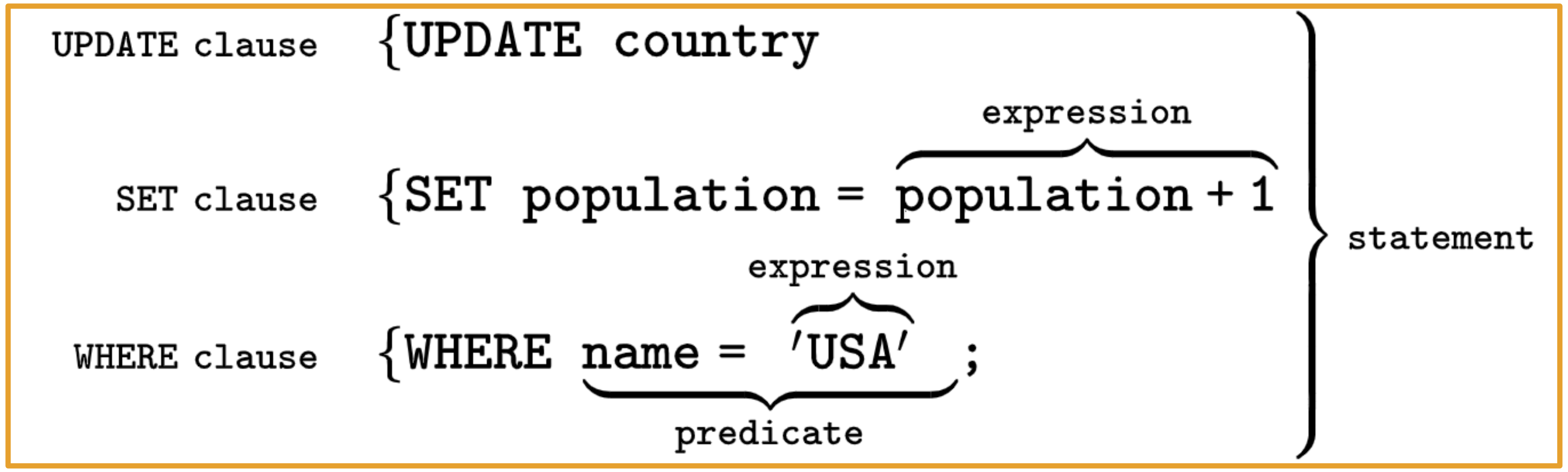
https://en.wikipedia.org/wiki/SQL

SQL was originally based upon relational algebra and tuple relational calculus. SQL is a declarative language. We say what data we want, not how to get it. We cannot manage <u>how</u> SQL obtains the data.

The scope of SQL includes data query, data manipulation (insert, update and delete), data definition (schema creation and modification), and data access control.

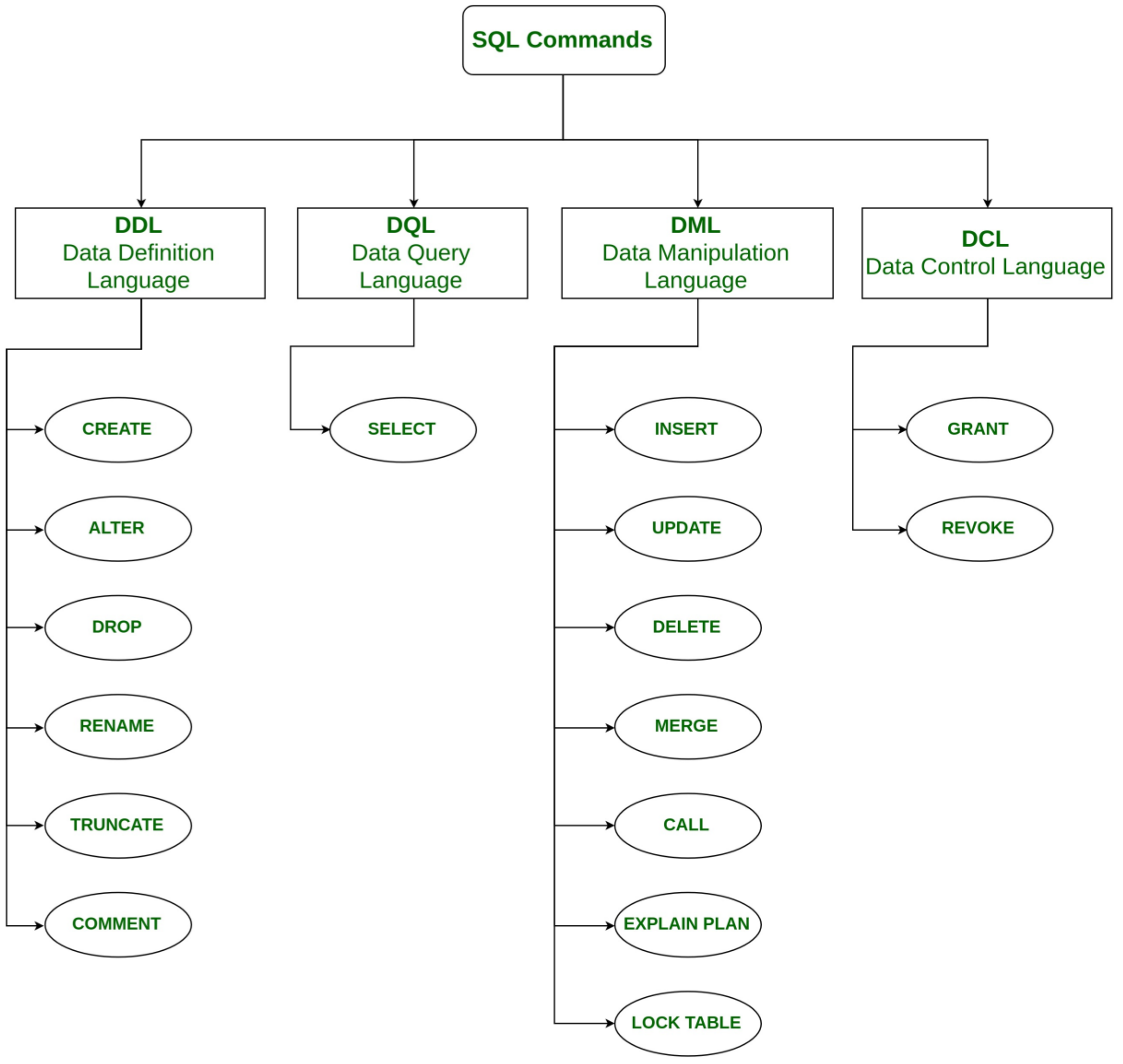
SQL consists of two main types of statements:

- Data Definition Language (DDL)
- Data Manipulation Language (DML).



```
UPDATE country
SET population = population + 1
WHERE name = 'USA' ;
```

## Types of SQL Commands



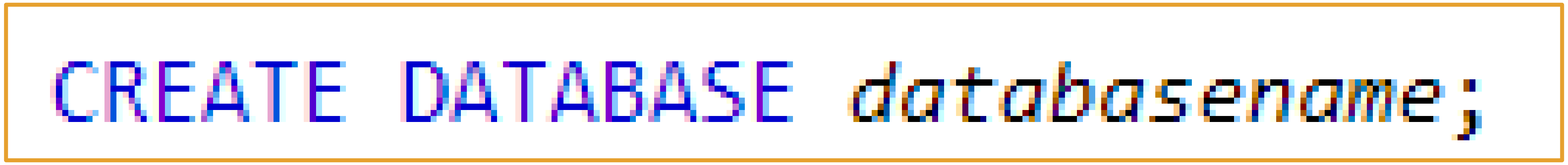
# Data Definition Language

https://docs.microsoft.com/en-us/sql/t-sql/statements/statements?view=sql-server-ver15#data-definition-language

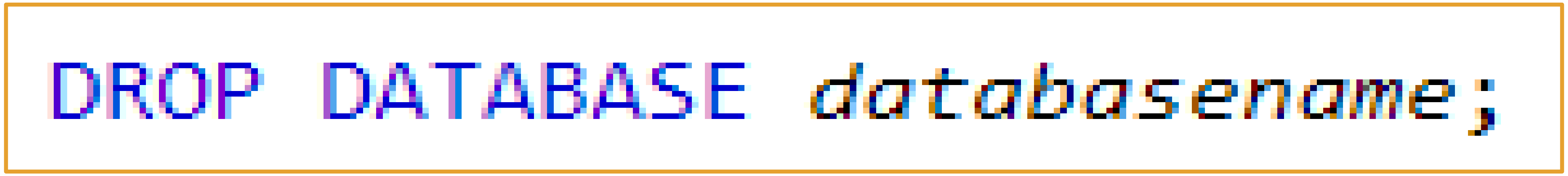
***Data Definition Language (DDL)*** statements define the structure of the DB. DDL statements create, alter, or drop the data <u>structures (tables)</u> of a database.

- ALTER - Modifies a table definition by altering, adding, or dropping columns and constraints.
- CREATE - creates a new database
- DROP - Removes one or more table definitions and all data, indexes, triggers, constraints, and permission specifications for those tables.

## SQL – ***Create*** and ***Drop*** an empty DB



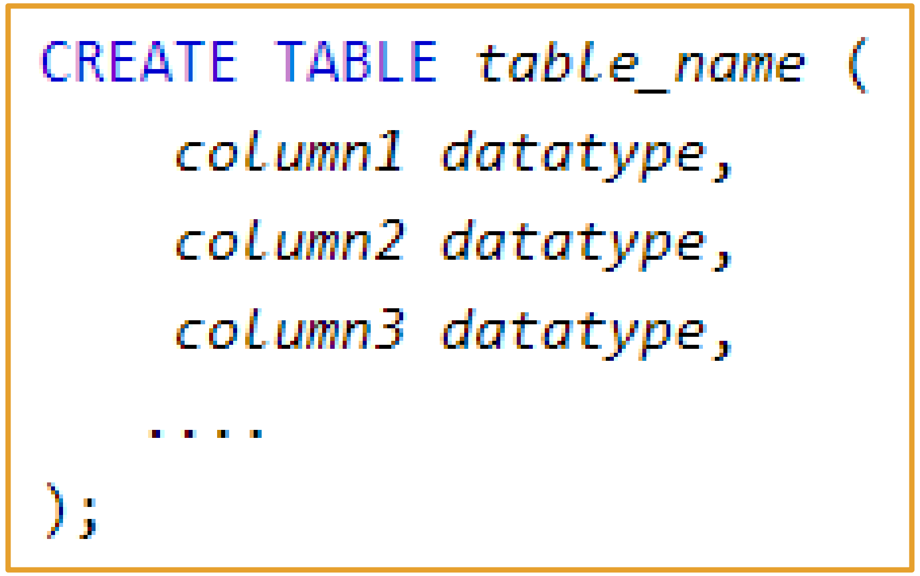
```
CREATE DATABASE databasename;
```



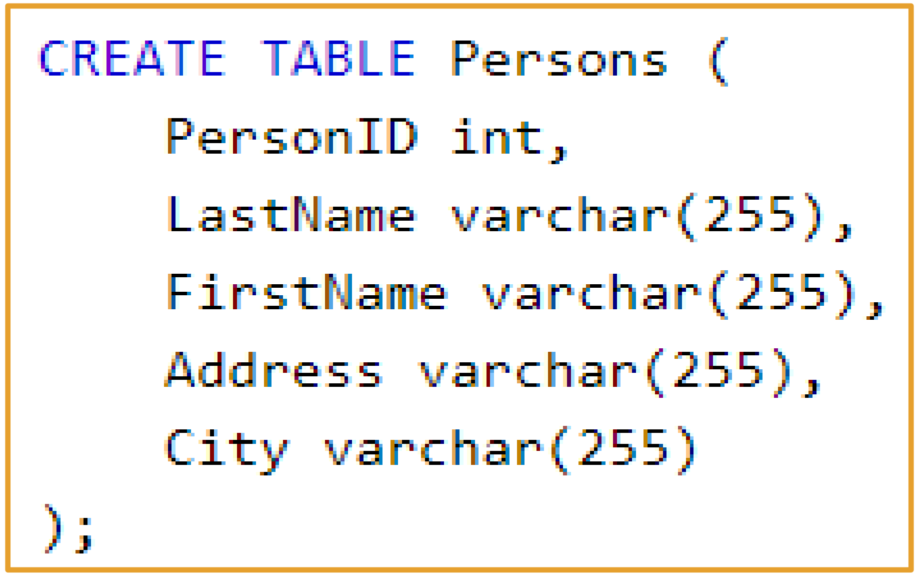
```
DROP DATABASE databasename;
```

*Deleting a database will result in complete loss of information stored in the database!

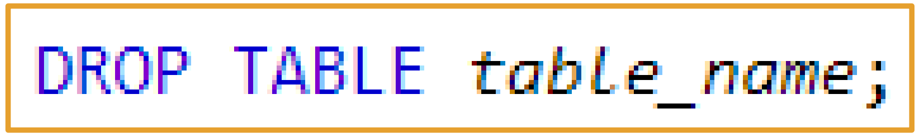
## SQL – ***Create*** and ***Drop*** a table



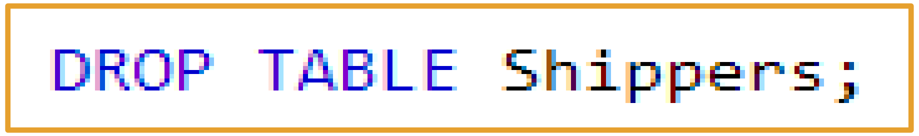
```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```



```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```



```
DROP TABLE table_name;
```

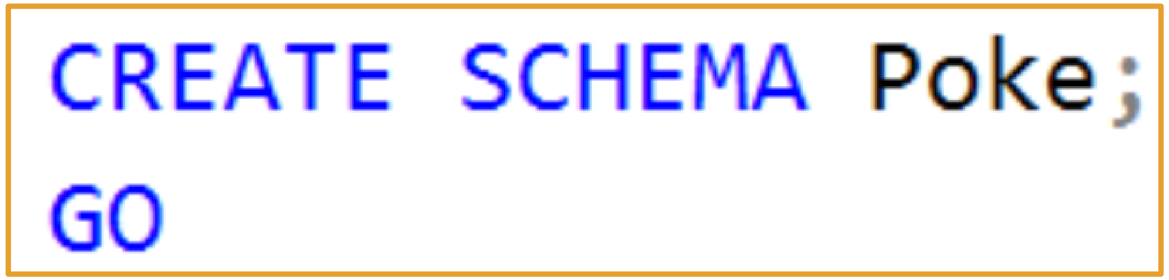


```
DROP TABLE Shippers;
```

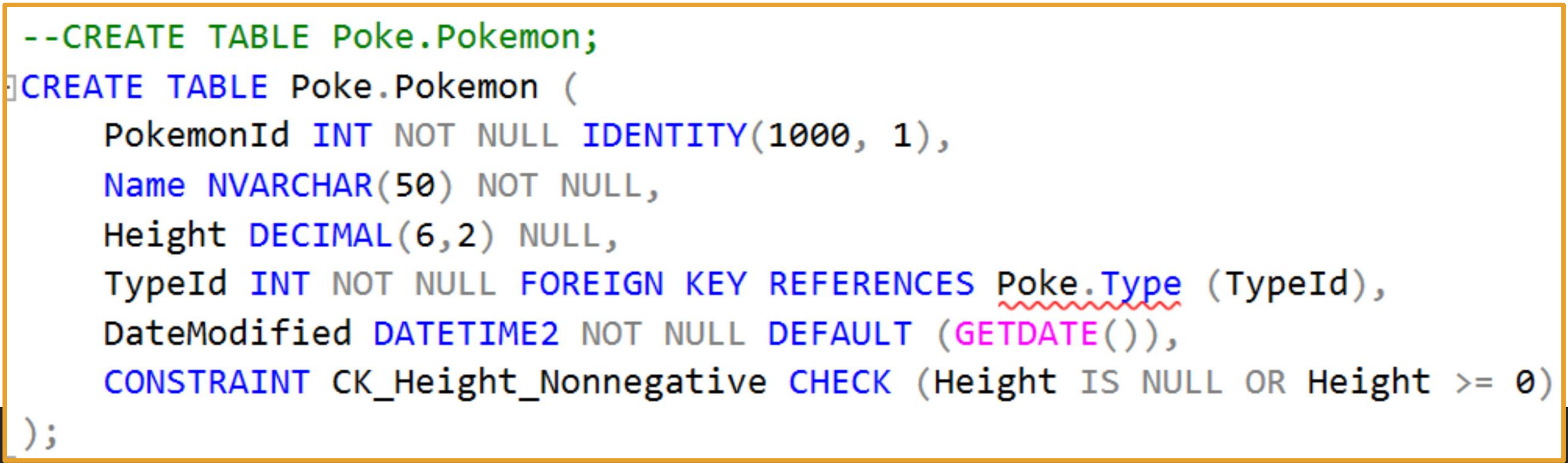
*Deleting a table will result in complete loss of information stored in the table!

# SQL with SQL Server

In SQL Server, every table must be in a schema.



```sql
CREATE SCHEMA Poke;
GO
```



```sql
--CREATE TABLE Poke.Pokemon;
CREATE TABLE Poke.Pokemon (
    PokemonId INT NOT NULL IDENTITY(1000, 1),
    Name NVARCHAR(50) NOT NULL,
    Height DECIMAL(6,2) NULL,
    TypeId INT NOT NULL FOREIGN KEY REFERENCES Poke.Type (TypeId),
    DateModified DATETIME2 NOT NULL DEFAULT (GETDATE()),
    CONSTRAINT CK_Height_Nonnegative CHECK (Height IS NULL OR Height >= 0)
);
```

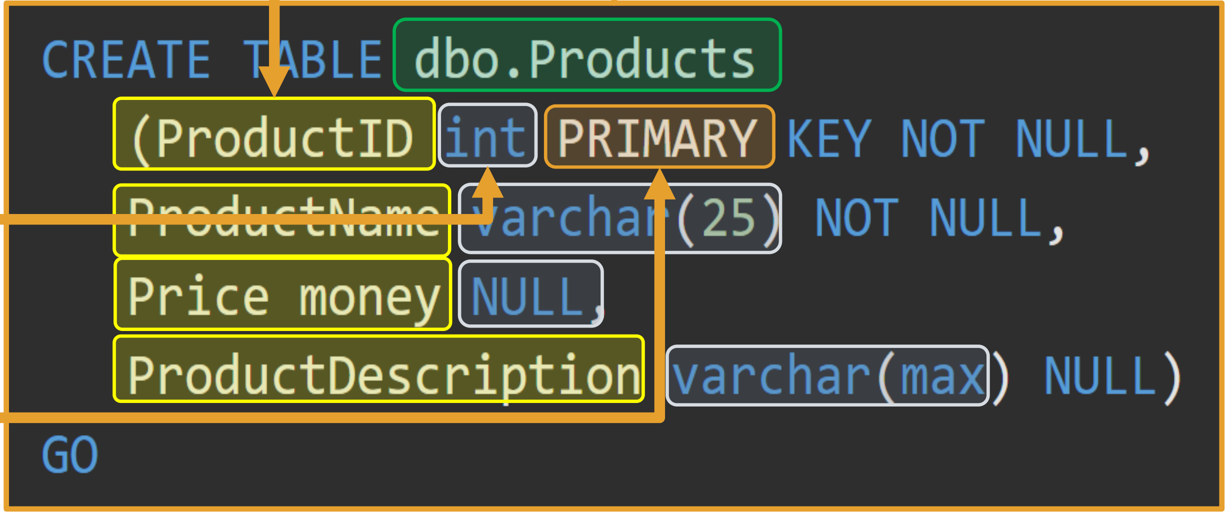
# Create a Table in SQL Server

https://docs.microsoft.com/en-us/sql/t-sql/lesson-1-creating-database-objects?view=sql-server-ver15#create-a-table

To create a table, you must provide:

1. a name for the table
2. Name of each column
3. data type of each column
4. a unique primary key.

```sql
CREATE TABLE dbo.Products
   (ProductID int PRIMARY KEY NOT NULL,
   ProductName varchar(25) NOT NULL,
   Price money NULL,
   ProductDescription varchar(max) NULL)
GO
```



```sql
CREATE TABLE dbo.Products
   (ProductID int PRIMARY KEY NOT NULL,
   ProductName varchar(25) NOT NULL,
   Price money NULL,
   ProductDescription varchar(max) NULL)
GO
```

# SQL – Attribute Constraints

https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-table-table-constraint-transact-sql?view=sql-server-ver15

| | |
|---|---|
| ***NOT NULL*** | column does not accept NULL as a value |
| ***NULL*** | column explicitly accepts NULL as a value. (NULL will be the default value) |
| ***PRIMARY KEY*** | value must be unique within this column |
| ***UNIQUE*** | implies NOT NULL and UNIQUE, and by default sets a ***CLUSTERED INDEX***. |
| ***FOREIGN KEY*** | by default sets a NONCLUSTERED INDEX |
| ***CHECK*** | enforces that some expression is true for every row |
| ***DEFAULT*** | configures a default value for that column |
| ***IDENTITY*** | this sets up an auto-incrementing default, AND prevents anyone from inserting their own value |

## SQL – String Data Types

https://docs.microsoft.com/en-us/sql/t-sql/data-types/char-and-varchar-transact-sql?view=sql-server-ver15

| Data Type | Description |
|---|---|
| CHAR(n) | Fixed-length up to n, 0 to 255, Default 1 |
| VARCHAR(n) | Variable length up to n. 0 to 65535 |
| NCHAR(n) | Fixed-length, Unicode string |
| NVARCHAR(n) | variable-length Unicode string. (Use this unless you nee to use something else) |

There are a many functions to deal with strings:

- LEN, SUBSTRING, CHARINDEX, REPLACE, LOWER, UPPER

## SQL – Integer Data Types

https://docs.microsoft.com/en-us/sql/t-sql/data-types/data-types-transact-sql?view=sql-server-ver15#exact-numerics

| Data Type | Description |
|---|---|
| BIT | It's a bit. 1 to 64. Default 1 |
| TINYINT | Signed = -128 to 127. unsigned = 0 to 255 |
| BOOL | Max 255 bytes. 0 = false, 1 = true |
| SMALLINT | Max  65535 bytes |
| INT | signed = 2147483648 to 2147483647. unsigned = 0 to 4294967295 (Use this unless you need something else) |

## SQL – Float Data Types

https://docs.microsoft.com/en-us/sql/t-sql/data-types/data-types-transact-sql?view=sql-server-ver15#approximate-numerics

| Data Type | Description |
|---|---|
| FLOAT(size,d) | size = total digits. d = number of digits after the decimal point |
| FLOAT(p) | If *p* is from 0 to 24, the data type becomes FLOAT(). If *p* is from 25 to 53, the data type becomes DOUBLE() |
| DOUBLE(size, d) | size = total digits. d = number of digits after the decimal point. |
| DECIMAL(*size*, *d*) | An exact fixed-point number. size = total digits(default 10, max 65). d = number of digits after the decimal point(default 0, max 30). |

# SQL – Date and Time Data Types

https://docs.microsoft.com/en-us/sql/t-sql/data-types/date-and-time-types?view=sql-server-ver15

| Data Type | Description |
|---|---|
| DATE | Format: YYYY-MM-DD. From '1000-01-01' to '9999-12-31' |
| DATETIME(fsp) | Format: YYYY-MM-DD hh:mm:ss. Add ***DEFAULT*** and ***ON UPDATE*** in the column definition to get automatic initialization and updating to the current date and time |
| TIMESTAMP(fsp) | The number of seconds since the Unix epoch. Format: YYYY-MM-DD hh:mm:ss. Automatic initialization and updating with DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition. |
| TIME(fsp) | hh:mm:ss. From '-838:59:59' to '838:59:59’ |
| YEAR | 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format. |
| DATETIMEOFFSET | For storing intervals of time. Use ***YEAR()*** to extract parts of the dates/times, DATEPART(YEAR FROM '2019-01-01') or DATEPART(YEAR, '2019-01-01') |

# SQL – Currency Data Types

https://docs.microsoft.com/en-us/sql/t-sql/data-types/data-types-transact-sql?view=sql-server-ver15#exact-numerics

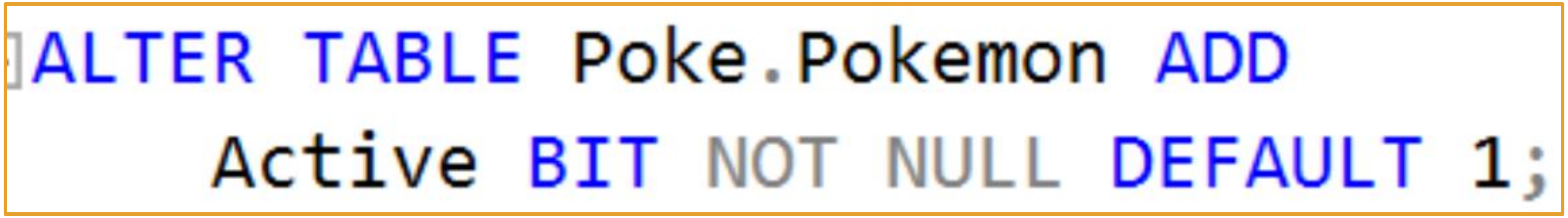
| Data Type | Description |
|---|---|
| MONEY | From -922,337,203,685,477.5808 to 922,337,203,685,477.5807 with ‘$’ |
| SMALLMONEY | From -214,748.3648 to 214,748.3647 with ‘$’ |

## ALTER Table

https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-table-transact-sql?view=sql-server-ver15

ALTER TABLE

- modifies a table definition by altering, adding, or dropping columns and constraints.
- reassigns and rebuilds partitions or disables and enables constraints and triggers.



```
ALTER TABLE Poke.Pokemon ADD
    Active BIT NOT NULL DEFAULT 1;
```

## AGGREGATE Functions

https://docs.microsoft.com/en-us/sql/t-sql/functions/aggregate-functions-transact-sql?view=sql-server-ver15
https://docs.microsoft.com/en-us/sql/t-sql/functions/functions?view=sql-server-ver15#aggregate-functions

Aggregate functions:

- perform a calculation on a set of values and returns a single value.
- ignore null values (except for ***COUNT()*** ).
- are often used with the ***GROUP BY*** clause of the ***SELECT*** statement.

These are Aggregate functions

| | |
|---|---|
| APPROX_COUNT_DISTINCT | MIN |
| AVG | STDEV |
| CHECKSUM_AGG | STDEVP |
| COUNT | STRING_AGG |
| COUNT_BIG | SUM |
| GROUPING | VAR |
| GROUPING_ID | VARP |
| MAX | |

# AVG( ) - Average

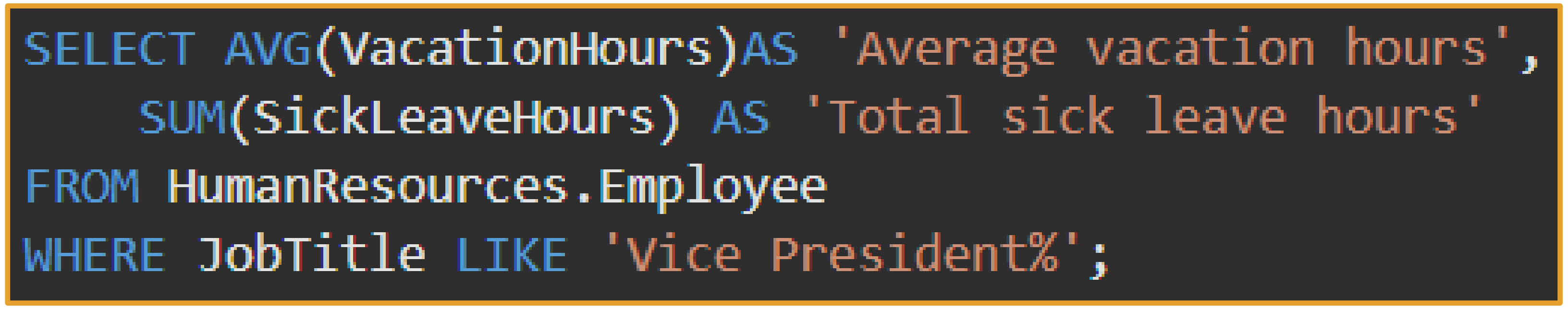
https://docs.microsoft.com/en-us/sql/t-sql/functions/avg-transact-sql?view=sql-server-ver15

***AVG( )*** computes the average of a set of values by dividing the sum of those values by the count of non-null values. If the sum exceeds the maximum value for the data type of the return value, ***AVG( )*** will return an error. ***AVG( )*** can have 1 or 2 arguments.

- ALL – (default) Applies the aggregate function to all values.
- DISTINCT - Specifies that ***AVG()*** operates only on one unique instance of each value, regardless of how many times that value occurs.

EX. **SELECT AVG(ALL NumbersColumn) FROM TableName;** returns the average of all numbers. Even duplicates.

This example returns the average vacation hours each Vice President has and how many total sick leave your all Vice Presidents have together.



```
SELECT AVG(VacationHours)AS 'Average vacation hours',
    SUM(SickLeaveHours) AS 'Total sick leave hours'
FROM HumanResources.Employee
WHERE JobTitle LIKE 'Vice President%';
```

# COUNT( )

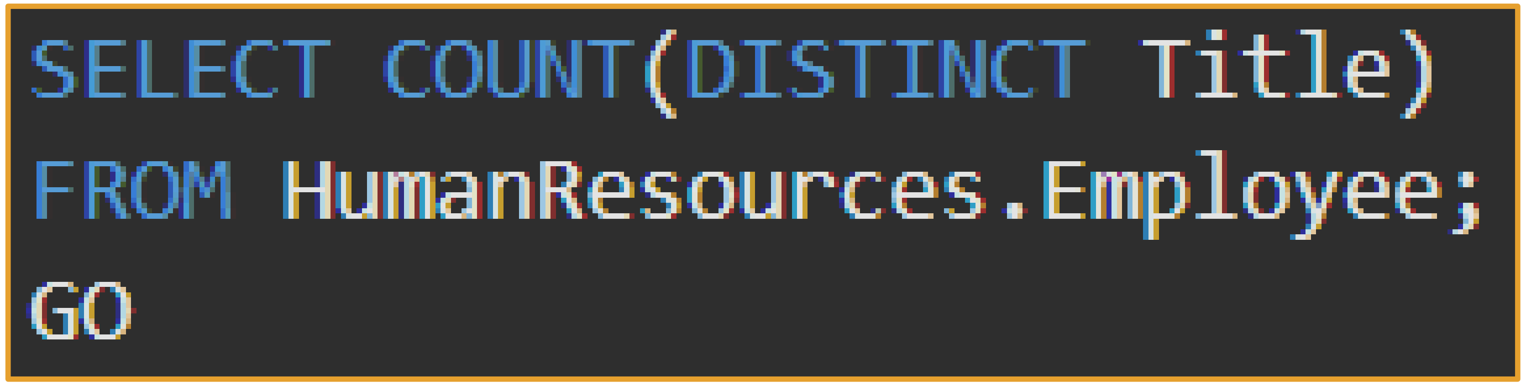
https://docs.microsoft.com/en-us/sql/t-sql/functions/count-transact-sql?view=sql-server-ver15

*COUNT( )* returns the number of items found in a group. *COUNT( )* always returns an ***int***.

*COUNT( )* has two possible arguments

- ALL - Applies the aggregate function to all values. ALL serves as the default.
- DISTINCT - Specifies that COUNT returns the number of unique nonnull values.

This example returns the number of unique job titles there are in all.



```
SELECT COUNT(DISTINCT Title)
FROM HumanResources.Employee;
GO
```

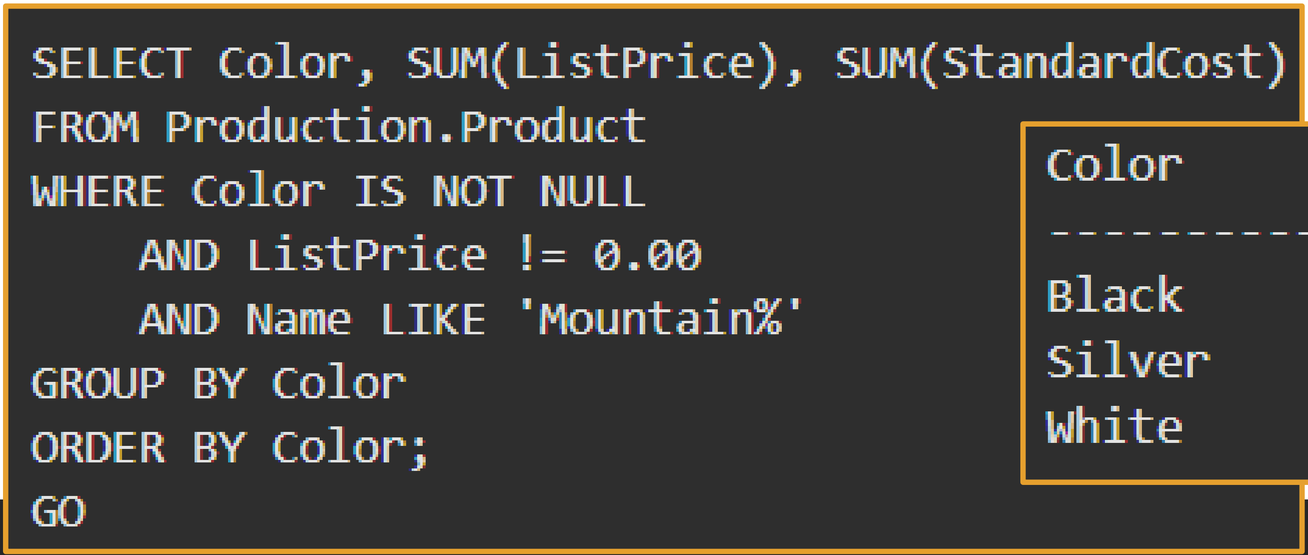
# SUM( )

https://docs.microsoft.com/en-us/sql/t-sql/functions/sum-transact-sql?view=sql-server-ver15

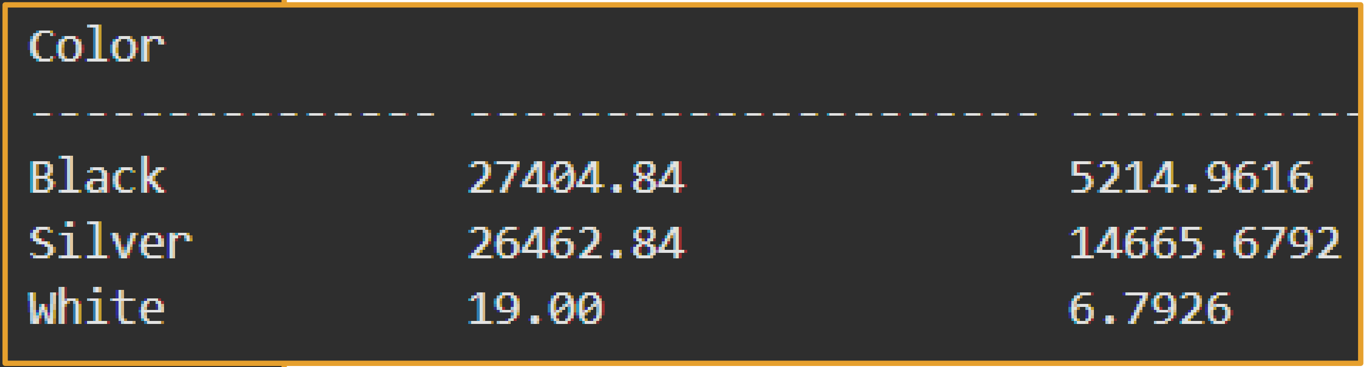
*SUM()* can be used with numeric columns only. Null values are ignored.

*SUM( )* has two possible arguments

- ALL – Default. Applies the aggregate function to all values.
- DISTINCT - Specifies that SUM returns the sum of unique values.



```
SELECT Color, SUM(ListPrice), SUM(StandardCost)
FROM Production.Product
WHERE Color IS NOT NULL
    AND ListPrice != 0.00
    AND Name LIKE 'Mountain%'
GROUP BY Color
ORDER BY Color;
GO
```



```
Color

--------------- ---------------------- ----------
Black           27404.84               5214.9616
Silver          26462.84               14665.6792
White           19.00                  6.7926
```