Beginning SQL

1. Installing SQL Server Management Studio
   1. SSMS
      1. <https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15>
   2. SQL Server
      1. <https://www.microsoft.com/en-us/sql-server/sql-server-downloads>
2. Creating Tables
   1. In order to create tables, you must create a database. Right-click database, new database.
   2. You can create tables in one of two ways: 1- You right click tables, new tables. 2-Crate tables with scripts:

CREATE TABLE EmployeeDemographics

(EmployeeID int,

FirstName varchar(50),

LastName varchar(50),

Age int,

Gender varchar(50)

Click the execute button

Refresh the tables

Ctrl-shift-r to clear the red swiggly error lines.

View tables by right-clicking table, show the first 1000 rows

1. Inserting Data

--INSERT INTO EmployeeDemographics VALUES

--(1002, 'Pam', 'Beaseley', 30, 'Female'),

--(1003, 'Dwight', 'Schrute', 29, 'Male'),

--(1004, 'Angela', 'Martin', 31, 'Female'),

--(1005, 'Toby', 'Flenderson', 32, 'Male'),

--(1006, 'Micheal', 'Scott', 35, 'Male'),

--(1007, 'Meredith', 'Palmer', 32, 'Female'),

--(1008, 'Stanley', 'Hudson', 38, 'Male'),

--(1009, 'Kevin', 'Malone', 31, 'Male')

INSERT INTO EmployeeSalary VALUES

(1001, 'Salesman', 45000),

(1002, 'Receptionist', 36000),

(1003, 'Salesman', 630000),

(1004, 'Accountant', 47000),

(1005, 'HR', 50000),

(1006, 'Regional Manager', 65000),

(1007, 'Supplier Ralations', 41000),

(1008, 'Salesman', 48000),

(1009, 'Accountant', 42000)

1. Select and From Statements

SELECT FirstName, LastName

* 1. From EmployeeDemographics;

SELECT \*

1. From EmployeeDemographics

SELECT TOP 5 \*

1. From EmployeeDemographics

SELECT DISTINCT(EmployeeID)

1. From EmployeeDemographics

SELECT Count(LastName) AS LastNameCount

1. From EmployeeDemographics

SELECT Count(LastName)

1. From EmployeeDemographics

SELECT MAX(Salary)

1. From EmployeeSalary;

SELECT MIN(Salary)

1. From EmployeeSalary;

SELECT AVG(Salary)

1. From EmployeeSalary;

SELECT COUNT(lastName) As lastNameCount

1. From EmployeeDemo;

SELECT \*

1. FROM [SQL Tutorial].dbo.EmployeeSalary
2. Where Statement
   1. SELECT \*

FROM EmployeeDemographics

Where FirstName = 'Jim'

SELECT \*

FROM EmployeeDemographics

1. Where FirstName <> 'Jim'
2. SELECT \*

FROM EmployeeDemographics

Where Age > 30

SELECT \*

FROM EmployeeDemographics

1. Where Age <= 32

SELECT \*

FROM EmployeeDemographics

1. Where Age <= 32 and Gender = 'Male';

SELECT \*

FROM EmployeeDemographics

1. Where Age <= 32 or Gender = 'Male';

SELECT \*

FROM EmployeeDemographics

Where LastName LIKE ‘$S$’

1. SELECT \*

FROM EmployeeDemographics

WHERE LastName LIKE 'S%'

1. SELECT \*

FROM EmployeeDemographics

WHERE LastName LIKE '%S%'

1. SELECT \*

FROM EmployeeDemographics

WHERE LastName LIKE 'S%o%'

1. SELECT \*

FROM EmployeeDemographics

WHERE FirstName is NOT NULL

1. SELECT \*

FROM EmployeeDemographics

WHERE FirstName IN ('Jim', 'Micheal')

1. GroupBy
   1. SELECT Gender

FROM EmployeeDemographics

GROUP BY Gender

* 1. SELECT Gender, Count(Gender) AS 'GenderCount'

FROM EmployeeDemographics

GROUP BY Gender

* 1. SELECT Gender, Count(Gender) AS 'GenderCount'

FROM EmployeeDemographics

Where Age > 31

GROUP BY Gender

* 1. SELECT Gender, Count(Gender) AS 'GenderCount'

FROM EmployeeDemographics

Where Age > 31

GROUP BY Gender

ORDER BY GenderCount DESC

* 1. SELECT \*

FROM EmployeeDemographics

ORDER BY Age DESC, Gender DESC

* 1. SELECT \*

FROM EmployeeDemographics

ORDER BY 4 DESC, 5 DESC

Intermediate SQL

* 1. Joins
     1. Inner Join

SELECT \*

FROM [SQL Tutorial].dbo.EmployeeDemographics

JOIN INNER [SQL Tutorial].dbo.EmployeeSalary

ON EmployeeDemographics.EmployeeID = EmployeeSalary.EmployeeID

* + 1. Full Outer Joins
       1. SELECT \*

FROM [SQL Tutorial].dbo.EmployeeDemographics

OUTER JOIN [SQL Tutorial].dbo.EmployeeSalary

ON EmployeeDemographics.EmployeeID = EmployeeSalary.EmployeeID

* + 1. Left Outer Join
       1. SELECT \*

FROM [SQL Tutorial].dbo.EmployeeDemographics

LEFT OUTER JOIN [SQL Tutorial].dbo.EmployeeSalary

ON EmployeeDemographics.EmployeeID = EmployeeSalary.EmployeeID

* + 1. Right Outer Join
       1. SELECT \*

FROM [SQL Tutorial].dbo.EmployeeDemographics

RIGHT OUTER JOIN [SQL Tutorial].dbo.EmployeeSalary

ON EmployeeDemographics.EmployeeID = EmployeeSalary.EmployeeID

* + - 1. SELECT EmployeeDemographics.EmployeeID, FirstName, LastName, JobTitle, Salary

FROM [SQL Tutorial].dbo.EmployeeDemographics

RIGHT OUTER JOIN [SQL Tutorial].dbo.EmployeeSalary

ON EmployeeDemographics.EmployeeID = EmployeeSalary.EmployeeID

* 1. Unions
     1. A join will join 2 table based on a common column
     2. A union will simply combine all columns in both tables
     3. SELECT \*

FROM [SQL Tutorial].dbo.EmployeeDemographics

Full Outer Join [SQL Tutorial].dbo.WareHouseEmployeeDemographics

ON EmployeeDemographics.EmployeeID =

WareHouseEmployeeDemographics.EmployeeID

* + 1. SELECT \*

FROM SQLTutorial.dbo.EmployeeDemographics

UNION

SELECT \*

FROM SQLTutorial.dbo.WareHouseEmployeeDemographics

* + 1. SELECT \*

FROM [SQL Tutorial].dbo.EmployeeDemographics

UNION ALL

SELECT \*

FROM [SQL Tutorial].dbo.WareHouseEmployeeDemographics

SELECT \*

FROM [SQL Tutorial].dbo.EmployeeDemographics

UNION ALL

SELECT \*

FROM [SQL Tutorial].dbo.WareHouseEmployeeDemographics

ORDER BY EmployeeID

Unions are only useful when the columns from both tables are the same.

* 1. Having vs. Group By Statements
  2. GETDATE()
  3. Primary Key vs. Primary Key
  4. Case Statements
     1. Note the syntax of the case statement and how similar it is to excel’s conditional statement, Case, When, Then, Else, End
     2. Nested Case Statements execute in consecutive order.

SELECT FirstName, LastName, Age,

CASE

When age > 30 then 'Old'

Else 'young'

END

FROM [SQL Tutorial].dbo.EmployeeDemographics

WHERE Age is not null

Order by age

* + 1. There’s a myriad amount logical operations than can be performed under the then statement of the case statement.

SELECT FirstName,LastName, JobTitle, Salary,

CASE

when jobtitle = 'salesman' then Salary + (Salary \* .10)

when jobtitle = 'accountant' then Salary + (Salary \* .05)

when jobtitle = 'HR' then Salary + (Salary \* .000001)

else salary + (Salary \* .003)

END AS Raise

FROM [SQL Tutorial].DBO.eMPLOYEEdEMOGRAPHICS

join [SQL Tutorial].dbo.EmployeeSalary

on EmployeeDemographics.employeeid = employeesalary.employeeid

1. Having Clause
   1. An aggregate can not appear in the Where clause like (below) but must be placed a having clause.

SELECT JobTitle, Count(Jobtitle)

FROM [SQL Tutorial].dbo.EmployeeDemographics

JOIN [SQL Tutorial].dbo.EmployeeSalary

ON EmployeeDemographics.EmployeeID = EmployeeSalary.EmployeeID

Where Count(Jobtitle) > 1

Group By Jobtitle

* 1. SELECT JobTitle, Count(Jobtitle)

FROM [SQL Tutorial].dbo.EmployeeDemographics

JOIN [SQL Tutorial].dbo.EmployeeSalary

ON EmployeeDemographics.EmployeeID = EmployeeSalary.EmployeeID

Group By Jobtitle

Having Count(Jobtitle) > 1

* + 1. This is how your query aggregate data after it has been aggregated by group by.
  1. SELECT JobTitle, Count(Jobtitle), AVG(Salary)

FROM [SQL Tutorial].dbo.EmployeeDemographics

JOIN [SQL Tutorial].dbo.EmployeeSalary

ON EmployeeDemographics.EmployeeID = EmployeeSalary.EmployeeID

Group By Jobtitle

Having Avg(Salary) > 45000

Order By AVG(Salary)

* + 1. The having clause goes between Group By And Order By

1. Updating and Deleting
   1. Updating

Update [SQL Tutorial].dbo.EmployeeDemographics

SET Age = '31', Gender = 'Female'

Where firstName = 'Holly' and lastname = 'Flask'

* 1. Delete

DELETE from [SQL Tutorial].dbo.EmployeeDemographics

Where EmployeeID = 1005

* + - 1. Delete statements are irreversible at execution, so use the select statement first to be sure.

1. Aliasing
   1. Alaising is for readability
   2. SELECT FirstName as fName

From [SQL Tutorial].dbo.EmployeeDemographics

* 1. SELECT FirstName + ' ' + LastName

From [SQL Tutorial].dbo.EmployeeDemographics

* 1. SELECT AVG(Age) as AvgAge

From [SQL Tutorial].dbo.EmployeeDemographics

* 1. Select Demo.EmployeeID

From [SQL Tutorial].dbo.[EmployeeDemographics] AS Demo

* 1. Select Demo.EmployeeID

From [SQL Tutorial].dbo.[EmployeeDemographics] AS Demo

JOIN [SQL Tutorial].dbo.[EmployeeSalary] AS Sal

ON Demo.EmployeeID = Sal.emp

1. Partition By
   1. Produces aggregated columns without rolling up other columns as in a Group By Statement.

SELECT FirstName, LastName, Gender, Salary,

Count(Gender) OVER (Partition By Gender) as TotalGender

From EmployeeDemographics Demo

Left Join EmployeeSalary Sal

ON Demo.EmployeeID = Sal.EmployeeID

* 1. The Group By Function Rolls columns up to the aggregated column.

SELECT Gender, Count(Gender)

From EmployeeDemographics Demo

Left Join EmployeeSalary Sal

ON Demo.EmployeeID = Sal.EmployeeID

Group By Gender

Advanced SQL

1. CTE’s (Common Table Expressions)
   1. Similar to a temporary tables and subqueries, a CTE provides a customized set of data from a database that you can query:

WITH CTE\_Employee as

(Select FirstName, LastName, Gender, Salary

, Count(gender) OVER (Partition by Gender) as TotalGender

, Avg(Salary) OVER (Partition By Gender) as AvgSalary

From [SQL Tutorial].dbo.EmployeeDemographics emp

Join [SQL Tutorial].dbo.EmployeeSalary sal

ON emp.EmployeeID = sal.EmployeeID

Where Salary > '45000'

)

Select \*

FROM CTE\_Employee

1. Temp Tables
   * 1. Intro
        1. Temp tables are more permanent then subquries
     2. Create Table

CREATE TABLE #temp\_employee (

EmployeeID int,

JobTitle varchar(100),

Salary int

* + 1. Insert Data

INSERT INTO #temp\_employee VALUES (

'1001', 'HR', '45000'

* + 1. Run

Select \*

From #temp\_employee

* + 1. Use Cases
       1. Use can create a temporary table from a common table, as a subset and run queries off that.
       2. Creating Joined Tables with any other conditions and placing them into a temp table so as to not have to keep reproducing the complexity of joining tables with conditions.
       3. Eventually, you’ll want to know about processing speed and storage in SQL

INSERT INTO #temp\_employee

SELECT \*

FROM [SQL Tutorial].dbo.EmployeeSalary

CREATE TABLE #Temp\_Employee2

JobTitle varchar(50),

EmployeePerJob int,

AvgAge int,

AvgSalary int)

SELECT \*

FROM #Temp\_Employee2

1. String Function(TRIM, LTRIMS, RTRIM, Replace, Substring, Upper, Lower)
   * 1. /\*
     2. Today's Topic: String Functions - TRIM, LTRIM, RTRIM, Replace, Substring, Upper, Lower
     3. \*/
     4. --Drop Table EmployeeErrors;
     5. CREATE TABLE EmployeeErrors (
     6. EmployeeID varchar(50)
     7. ,FirstName varchar(50)
     8. ,LastName varchar(50)
     9. )
     10. Insert into EmployeeErrors Values
     11. ('1001 ', 'Jimbo', 'Halbert')
     12. ,(' 1002', 'Pamela', 'Beasely')
     13. ,('1005', 'TOby', 'Flenderson - Fired')
     14. Select \*
     15. From EmployeeErrors
     16. -- Using Trim, LTRIM, RTRIM
     17. Select EmployeeID, TRIM(employeeID) AS IDTRIM
     18. FROM EmployeeErrors
     19. Select EmployeeID, RTRIM(employeeID) as IDRTRIM
     20. FROM EmployeeErrors
     21. Select EmployeeID, LTRIM(employeeID) as IDLTRIM
     22. FROM EmployeeErrors

* + 1. -- Using Replace
    2. Select LastName, REPLACE(LastName, '- Fired', '') as LastNameFixed
    3. FROM EmployeeErrors
    4. -- Using Substring
    5. Select Substring(err.FirstName,1,3), Substring(dem.FirstName,1,3), Substring(err.LastName,1,3), Substring(dem.LastName,1,3)
    6. FROM EmployeeErrors err
    7. JOIN EmployeeDemographics dem
    8. on Substring(err.FirstName,1,3) = Substring(dem.FirstName,1,3)
    9. and Substring(err.LastName,1,3) = Substring(dem.LastName,1,3
    10. -- Using UPPER and lower
    11. Select firstname, LOWER(firstname)
    12. from EmployeeErrors
    13. Select Firstname, UPPER(FirstName)
    14. from EmployeeErrors

1. Stored Procedures
   1. A group of SQL statement that can be created and stored in that database.
   2. Stored Procedures can accept input parameters which allows different users to use that same procedures with different data.
      1. CREATE PROCEDURE TEST
      2. AS
      3. SELECT \*
      4. FROM EmployeeDemographics
      5. EXEC TEST
      6. CREATE PROCEDURE Temp\_Employees
      7. AS
      8. CREATE TABLE #temp\_employee (
      9. JobTitle varchar(100),
      10. EmployeePerJob int,
      11. AvgAge int,
      12. AvgSalary int
      13. )
      14. INSERT INTO #temp\_employee
      15. SELECT JobTitle, Count(Jobtitle), Avg(Age), Avg(salary)
      16. FROM EmployeeDemographics emp
      17. JOIN EmployeeSalary sal
      18. ON emp.EmployeeID =sal.EmployeeID
      19. group by JobTitle
      20. SELECT \*
      21. FROM #temp\_employee
   3. ALTER\_PROCEDURE [dbo]
      1. EXEC Temp\_Employees
   4. Importing Data from different file Types/Source
   5. Exporting Data to different File Types
2. Subqueries
   1. Create Subquery
      1. You can put a subquery in the select statement
      2. Select EmployeeID, Salary, (Select Avg(Salary) From EmployeeSalary)

From Employee Salary

* 1. Partition By
     1. The Partition By statement statement below produced the exact same result
     2. Select EmployeeID, Salary, Avg(Salary) over () AllAvgSalary

From EmployeeSalary

* 1. Comparison
     1. By Comparison the group by statement will produce the same output because it will not produce the right result for Avg(Salary)
     2. Select EmployeeID, Salary, Avg(Salary) as allAvgSalary

From EmployeeSalary

Group By EmployeeID, Salary

Order By 1,2

* 1. From
     1. This is a subquery in the From statement

Select a.EmployeeID, AllAvgSalary

From (Select EmployeeID, Salary, Avg(Salary) over () AllAvgSalary

From EmployeeSalary) a

1. Subquery
   * 1. Select EmployeeID, JobTitle, Salary

From EmployeeSalary

1. Example

Select EmployeeID, JobTitle, Salary

From EmployeeSalary