Table of Contents

[SQL NOTES 1](#_Toc176845256)

[Creating, altering and dropping a database 1](#_Toc176845257)

[Creating and working with tables 2](#_Toc176845258)

[Adding Default Constraints to Table 3](#_Toc176845259)

[Cascading referential integrity 3](#_Toc176845260)

[Check Constraint 5](#_Toc176845261)

[Identity Column 5](#_Toc176845262)

[How to get the last generated identity column value in SQL Server 6](#_Toc176845263)

[Unique Key Constraint 6](#_Toc176845264)

[Select Statement in C# 7](#_Toc176845265)

[Group By 7](#_Toc176845266)

[Joins 10](#_Toc176845267)

[Self-Join 16](#_Toc176845268)

[**Coalesce() function** 17](#_Toc176845269)

[Union and Union All 18](#_Toc176845270)

# SQL NOTES

## Creating, altering and dropping a database

**A SQL Server database can be created, altered and dropped**  
1. Graphically using SQL Server Management Studio (SSMS) or  
2. Using a Query

**To create the database graphically**  
1. Right Click on Databases folder in the Object explorer  
2. Select New Database  
3. In the New Database dialog box, enter the Database name and click OK.  
**To Create the database using a query**  
Create database DatabaseName  
**Whether, you create a database graphically using the designer or, using a query, the following 2 files gets generated.**  
.MDF file - Data File (Contains actual data)  
.LDF file - Transaction Log file (Used to recover the database)

**To alter a database, once it's created**  
Alter database DatabaseName Modify Name = NewDatabaseName  
**Alternatively, you can also use system stored procedure**  
Execute sp\_renameDB 'OldDatabaseName','NewDatabaseName'  
  
**To Delete or Drop a database**  
Drop Database DatabaseThatYouWantToDrop

**Dropping a database, deletes the LDF and MDF files.**  
  
You cannot drop a database, if it is currently in use. You get an error stating - Cannot drop database "NewDatabaseName" because it is currently in use. So, if other users are connected, you need to put the database in single user mode and then drop the database.  
Alter Database DatabaseName Set SINGLE\_USER With Rollback Immediate  
With Rollback Immediate option, will rollback all incomplete transactions and closes the connection to the database.  
  
***Note: System databases cannot be dropped.***

## Creating and working with tables

**To create tblPerson table, graphically, using SQL Server Management Studio**  
**1.** Right click on Tables folder in Object explorer window  
**2.** Select New Table  
**3.** Fill Column Name, Data Type and Allow Nulls, as shown below and save the table as tblPerson.

The following statement creates tblGender table, with ID and Gender columns. **ID**column, is the **primary key** column. The primary key is used to uniquely identify each row in a table. Primary key does not allow nulls.  
Create Table **tblGender**  
(ID int Not Null Primary Key,  
Gender nvarchar(50))

**To graphically add a foreign key reference**  
1. Right click tblPerson table and select Design  
2. In the table design window, right click on GenderId column and select Relationships  
3. In the Foreign Key Relationships window, click Add button

4. Now expand, in Tables and Column Specification row, by clicking the, + sign  
5. Click on the elipses button, that is present in Tables and Column Specification row  
6. From the Primary Key Table, dropdownlist, select tblGender  
7. Click on the row below, and select ID column  
8. From the column on the right hand side, select GenderId  
9. Click OK and then click close.  
10. Finally save the table.

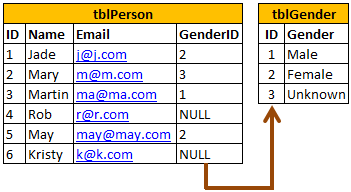
**To add a foreign key reference using a query**  
**Alter table** **tblPerson**   
**add constraint** **tblPerson\_GenderId\_FK FOREIGN KEY** **(GenderId) references tblGender(ID)**  
  
  
**The general formula is here**  
**Alter table ForeignKeyTable add constraint ForeignKeyTable\_ForiegnKeyColumn\_FK**  
**FOREIGN KEY (ForiegnKeyColumn) references PrimaryKeyTable (PrimaryKeyColumn)**  
  
  
**Foreign keys** are used to enforce **database integrity**. In layman's terms, A **foreign key** in one table points to a **primary key** in another table. The foreign key constraint prevents invalid data form being inserted into the foreign key column. The values that you enter into the foreign key column, has to be one of the values contained in the table it points to.

## Adding Default Constraints to Table

The default constraint is used to insert a default value into a column. The default value will be added to all new records, if no other value is specified, including NULL.  
  
**Altering an existing column to add a default constraint:**  
ALTER TABLE { TABLE\_NAME }  
ADD CONSTRAINT { CONSTRAINT\_NAME }  
DEFAULT { DEFAULT\_VALUE } FOR { EXISTING\_COLUMN\_NAME }  
  
**Adding a new column, with default value, to an existing table:**  
ALTER TABLE { TABLE\_NAME }   
ADD { COLUMN\_NAME } { DATA\_TYPE } { NULL | NOT NULL }   
CONSTRAINT { CONSTRAINT\_NAME } DEFAULT { DEFAULT\_VALUE }

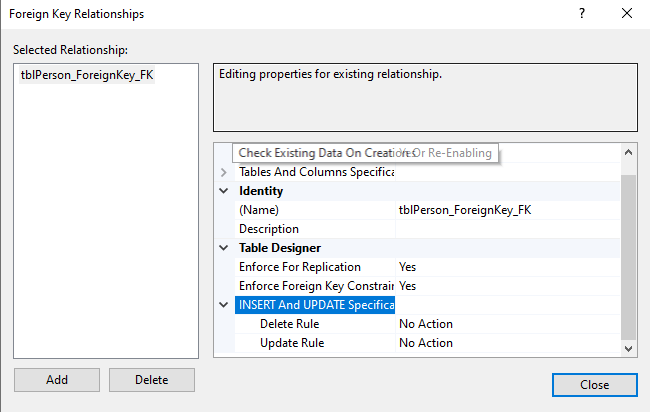
**The following command will add a default constraint, DF\_tblPerson\_GenderId.**  
ALTER TABLE tblPerson  
ADD CONSTRAINT DF\_tblPerson\_GenderId  
DEFAULT 1 FOR GenderId  
  
  
The insert statement below does not provide a value for GenderId column, so the default of 1 will be inserted for this record.  
Insert into tblPerson(ID,Name,Email) values(5,'Sam','s@s.com')  
  
On the other hand, the following insert statement will insert NULL, instead of using the default.  
Insert into tblPerson(ID,Name,Email,GenderId) values (6,'Dan','d@d.com',NULL)  
  
**To drop a constraint**  
ALTER TABLE { TABLE\_NAME }   
DROP CONSTRAINT { CONSTRAINT\_NAME }

## Cascading referential integrity

Cascading referential integrity constraint allows to define the actions Microsoft SQL Server should take when a user attempts to delete or update a key to which an existing foreign keys points.  
  
**For example**, consider the 2 tables shown below. If you delete row with **ID = 1** from **tblGender**table, then row with **ID = 3** from **tblPerson**table becomes an **orphan record**. You will not be able to tell the Gender for this row. So, Cascading referential integrity constraint can be used to define actions Microsoft SQL Server should take when this happens. By default, we get an error and the DELETE or UPDATE statement is rolled back.  
  


**However, you have the following options when setting up Cascading referential integrity constraint**  
**1. No Action**: This is the default behaviour. No Action specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, an error is raised and the DELETE or UPDATE is rolled back.  
**2. Cascade**: Specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are also deleted or updated.  
**3. Set NULL**: Specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are set to NULL.    
**4. Set Default**: Specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are set to default values.

We need to go to primary table where foreign key is referred to change this action:



## Check Constraint

* **CHECK constraint** is used to **limit the range of the values**, that can be entered for a column.
* Let's say, we have an integer AGE column, in a table. The AGE in general cannot be less than ZERO and at the same time cannot be greater than 150. But, since AGE is an integer column it can accept negative values and values much greater than 150. So, to limit the values, that can be added, we can use CHECK constraint. In SQL Server, CHECK constraint can be created graphically, or using a query.
* **The following check constraint, limits the age between ZERO and 150.**  
  ALTER TABLE tblPerson  
  ADD CONSTRAINT CK\_tblPerson\_Age CHECK (Age > 0 AND Age < 150)
* **The general formula for adding check constraint in SQL Server:**  
  ALTER TABLE { TABLE\_NAME }  
  ADD CONSTRAINT { CONSTRAINT\_NAME } CHECK ( BOOLEAN\_EXPRESSION )
* If the BOOLEAN\_EXPRESSION returns true, then the CHECK constraint allows the value, otherwise it doesn't. Since, AGE is a nullable column, it's possible to pass null for this column, when inserting a row. When you pass NULL for the AGE column, the boolean expression evaluates to **UNKNOWN**, and allows the value.
* **To drop the CHECK constraint:**  
  ALTER TABLE tblPerson  
  DROP CONSTRAINT CK\_tblPerson\_Age

## Identity Column

* If a column is marked as an identity column, then the values for this column are automatically generated, when you insert a new row into the table.
* The following, create table statement marks PersonId as an identity column with seed = 1 and Identity Increment = 1. Seed and Increment values are optional. If you don't specify the identity and seed they both default to 1

**Create Table tblPerson  
(  
PersonId int Identity(1,1) Primary Key,  
Name nvarchar(20)  
)**

* **In the following 2 insert statements, we only supply values for Name column and not for PersonId column.**  
  **Insert into tblPerson values ('Sam')  
  Insert into tblPerson values ('Sara')**  
  *If you select all the rows from tblPerson table, you will see that, 'Sam' and 'Sara' rows have got 1 and 2 as PersonId.*
* Now, if I try to execute the following query, I get an error stating - An explicit value for the identity column in table 'tblPerson' can only be specified when a column list is used and IDENTITY\_INSERT is ON.   
  **Insert into tblPerson values (1,'Todd')**
* So if you mark a column as an Identity column, you dont have to explicitly supply a value for that column when you insert a new row. The value is automatically calculated and provided by SQL server. So, to insert a row into tblPerson table, just provide value for Name column.  
  Insert into tblPerson values ('Todd')
* Delete the row, that you have just inserted and insert another row. You see that the value for PersonId is 2. Now if you insert another row, PersonId is 3. A record with PersonId = 1, does not exist, and I want to fill this gap. To do this, we should be able to explicitly supply the value for identity column. To explicitly supply a value for identity column
* First turn on identity insert - **SET Identity\_Insert tblPerson ON**
* In the insert query specify the column list  
      **Insert into tblPerson(PersonId, Name) values(2, 'John')**
* After, you have the gaps in the identity column filled, and if you wish SQL server to calculate the value, turn off Identity\_Insert.  
  **SET Identity\_Insert tblPerson OFF**
* If you have deleted all the rows in a table, and you want to reset the identity column value, use DBCC CHECKIDENT command. This command will reset PersonId identity column.  
  **DBCC CHECKIDENT(tblPerson, RESEED, 0)**

## How to get the last generated identity column value in SQL Server

* There are several ways in sql server, to retrieve the last identity value that is generated. The most common way is to use SCOPE\_IDENTITY() built in function.
* Apart, from using SCOPE\_IDENTITY(), you also have @@IDENTITY and IDENT\_CURRENT('TableName') function.
* Example queries for getting the last generated identity value  
  **Select SCOPE\_IDENTITY()  
  Select @@IDENTITY  
  Select IDENT\_CURRENT('tblPerson')**
* Let's say, I have 2 tables tblPerson1 and tblPerson2, and I have a trigger on tblPerson1 table, which will insert a record into tblPerson2 table. Now, when you insert a record into tblPerson1 table,
* SCOPE\_IDENTITY() returns the last identity value that is created in the same session (Connection) and in the same scope (in the same Stored procedure, function, trigger). SCOPE\_IDENTITY() returns the idetentity value that is generated in tblPerson1 table.
* @@IDENTITY returns, the value that is generated in tblPerson2 table. o, @@IDENTITY returns the last identity value that is created in the same session without any consideration to the scope.
* IDENT\_CURRENT('tblPerson') returns the last identity value created for a specific table across any session and any scope.

## Unique Key Constraint

We use UNIQUE constraint to enforce uniqueness of a column i.e the column shouldn't allow any duplicate values. We can add a Unique constraint thru the designer or using a query.  
**To add a unique constraint using SQL server management studio designer:**  
1. Right-click on the table and select Design  
2. Right-click on the column, and select Indexes/Keys...  
3. Click Add  
4. For Columns, select the column name you want to be unique.  
5. For Type, choose Unique Key.  
6. Click Close, Save the table.

To create the unique key using a query: **Alter Table Table\_Name  
Add Constraint Constraint\_Name Unique(Column\_Name)**Both primary key and unique key are used to enforce, the uniqueness of a column. So, when do you choose one over the other?  
A table can have, only one primary key. If you want to enforce uniqueness on 2 or more columns, then we use unique key constraint. **What is the difference between Primary key constraint and Unique key constraint? This question is asked very frequently in interviews.**1. A table can have only one primary key, but more than one unique key  
2. Primary key does not allow nulls, where as unique key allows one nullTo drop the constraint1. Right click the constraint and delete.  
Or  
2. Using a query **Alter Table tblPerson  
Drop COnstraint UQ\_tblPerson\_Email**

## Select Statement in C#

* **Basic select statement syntax**  
  SELECT Column\_List  
  FROM Table\_Name
* **If you want to select all the columns, you can also use \*. For better performance use the column list, instead of using \*.**  
  SELECT \*  
  FROM Table\_Name
* **To Select distinct rows use DISTINCT keyword**  
  SELECT DISTINCT Column\_List  
  FROM Table\_Name
* **Filtering rows with WHERE clause**  
  SELECT Column\_List  
  FROM Table\_Name  
  WHERE Filter\_Condition

Some Other Operators

* **IN**: Checks if a value is within a set of values.

SELECT \* FROM Employees WHERE Department IN ('Sales', 'Marketing');

* **BETWEEN**: Checks if a value is within a range.

SELECT \* FROM Employees WHERE Salary BETWEEN 30000 AND 50000;

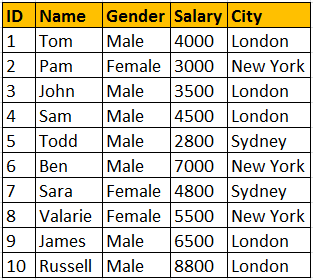
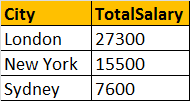
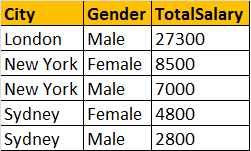
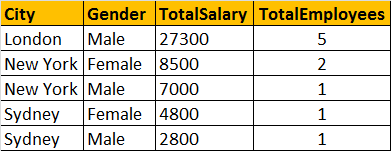
* **LIKE**: Searches for a specified pattern.

SELECT \* FROM Employees WHERE Name LIKE 'J%'; -- Names starting with 'J'

* **IS NULL**: Checks if a value is NULL.

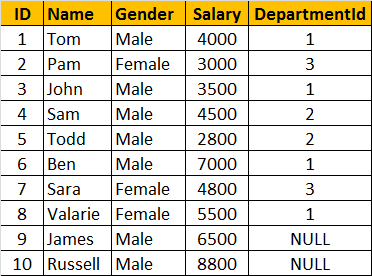
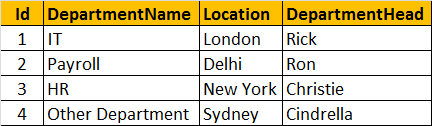
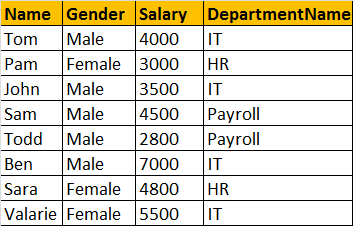
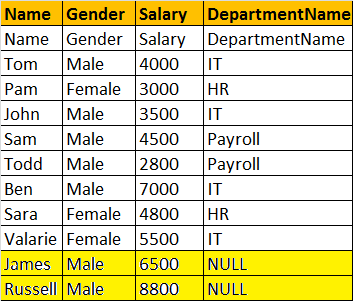
SELECT \* FROM Employees WHERE ManagerID IS NULL;

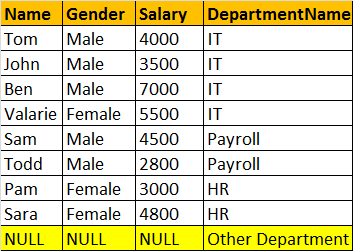
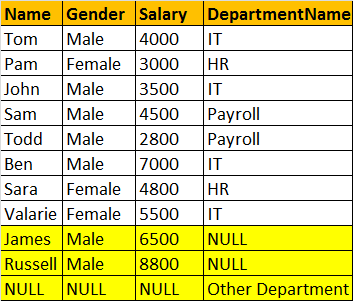
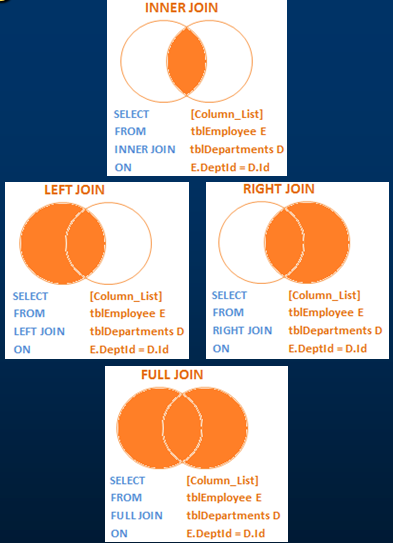
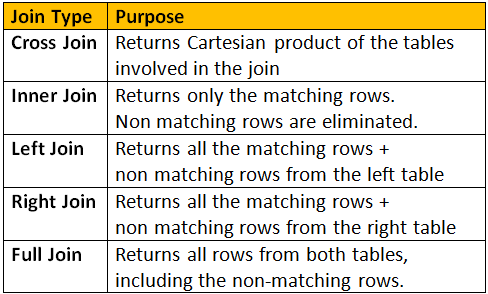
## Group By

In SQL Server we have got lot of aggregate functions. Examples  
1. Count()  
2. Sum()  
3. avg()  
4. Min()  
5. Max()  
**Group by** clause is used to group a selected set of rows into a set of summary rows by the values of one or more columns or expressions. It is always used in conjunction with one or more aggregate functions.  
  
  
I want an sql query, which gives total salaries paid by City. The output should be as shown below.  
  
  
**Query for retrieving total salaries by city**:  
We are applying SUM() aggregate function on Salary column, and grouping by city column. This effectively adds, all salaries of employees with in the same city.  
**Select City, SUM(Salary) as TotalSalary**  
**from tblEmployee**  
**Group by City**  
  
**Note:** If you omit, the group by clause and try to execute the query, you get an error - Column 'tblEmployee.City' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.   
  
Now, I want an sql query, which gives total salaries by City, by gender. The output should be as shown below.  
  
  
  
**Query for retrieving total salaries by city and by gender**: It's possible to group by multiple columns. In this query, we are grouping first by city and then by gender.   
**Select City, Gender, SUM(Salary) as TotalSalary**  
**from tblEmployee**  
**group by City, Gender**  
  
Now, I want an sql query, which gives total salaries and total number of employees by City, and by gender. The output should be as shown below.  
  
  
  
**Query for retrieving total salaries and total number of employees by City, and by gender**: The only difference here is that, we are using Count() aggregate function.  
**Select City, Gender, SUM(Salary) as TotalSalary,  
COUNT(ID) as TotalEmployees**  
**from tblEmployee**  
**group by City, Gender**  
  
**Filtering Groups:**  
WHERE clause is used to filter rows before aggregation, where as HAVING clause is used to filter groups after aggregations. The following 2 queries produce the same result.  
  
Filtering rows using WHERE clause, before aggrgations take place:  
**Select City, SUM(Salary) as TotalSalary**  
**from tblEmployee**  
**Where City = 'London'**  
**group by City**  
  
Filtering groups using HAVING clause, after all aggrgations take place:  
**Select City, SUM(Salary) as TotalSalary**  
**from tblEmployee**  
**group by City**  
**Having City = 'London'**  
  
From a performance standpoint, you cannot say that one method is less efficient than the other. Sql server optimizer analyzes each statement and selects an efficient way of executing it. As a best practice, use the syntax that clearly describes the desired result. Try to eliminate rows that   
you wouldn't need, as early as possible.  
  
**It is also possible to combine WHERE and HAVING**  
**Select City, SUM(Salary) as TotalSalary**  
**from tblEmployee**  
**Where Gender = 'Male'**  
**group by City**  
**Having City = 'London'**  
  
**Difference between WHERE and HAVING clause:**  
1. WHERE clause can be used with - Select, Insert, and Update statements, where as HAVING clause can only be used with the Select statement.  
2. WHERE filters rows before aggregation (GROUPING), where as, HAVING filters groups, after the aggregations are performed.  
3. Aggregate functions cannot be used in the WHERE clause, unless it is in a sub query contained in a HAVING clause, whereas, aggregate functions can be used in Having clause.

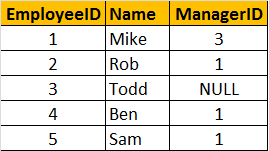
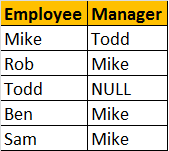
## Joins

**Joins in SQL server** are used to query (retrieve) data from 2 or more related tables. In general tables are related to each other using foreign key constraints.

**In SQL server, there are different types of JOINS.**  
1. CROSS JOIN  
2. INNER JOIN   
3. OUTER JOIN   
  
**Outer Joins are again divided into 3 types**  
1. Left Join or Left Outer Join  
2. Right Join or Right Outer Join  
3. Full Join or Full Outer Join  
  
**Now let's understand all the JOIN types, with examples and the differences between them.**  
**Employee Table (tblEmployee)**  
  
  
**Departments Table (tblDepartment)**  
  
**General Formula for Joins**  
SELECT      ColumnList  
FROM           LeftTableName  
JOIN\_TYPE  RightTableName  
ON                 JoinCondition  
  
**CROSS JOIN**  
CROSS JOIN, produces the cartesian product of the 2 tables involved in the join. For example, in the Employees table we have 10 rows and in the Departments table we have 4 rows. So, a cross join between these 2 tables produces 40 rows. Cross Join shouldn't have ON clause.   
  
**CROSS JOIN Query:**  
SELECT Name, Gender, Salary, DepartmentName  
FROM tblEmployee  
CROSS JOIN tblDepartment  
  
**JOIN or INNER JOIN**  
Write a query, to retrieve Name, Gender, Salary and DepartmentName from Employees and Departments table. The output of the query should be as shown below.  
  
  
SELECT Name, Gender, Salary, DepartmentName  
FROM tblEmployee  
INNER JOIN tblDepartment  
ON tblEmployee.DepartmentId = tblDepartment.Id  
**OR**  
SELECT Name, Gender, Salary, DepartmentName  
FROM tblEmployee  
JOIN tblDepartment  
ON tblEmployee.DepartmentId = tblDepartment.Id  
  
**Note:** JOIN or INNER JOIN means the same. It's always better to use INNER JOIN, as this explicitly specifies your intention.  
If you look at the output, we got only 8 rows, but in the Employees table, we have 10 rows. We didn't get JAMES and RUSSELL records. This is because the DEPARTMENTID, in Employees table is NULL for these two employees and doesn't match with ID column in Departments table.  
So, in summary, INNER JOIN, returns only the matching rows between both the tables. Non matching rows are eliminated.  
  
**LEFT JOIN or LEFT OUTER JOIN**  
Now, let's say, I want all the rows from the Employees table, including JAMES and RUSSELL records. I want the output, as shown below.  
  
SELECT Name, Gender, Salary, DepartmentName  
FROM tblEmployee  
LEFT OUTER JOIN tblDepartment  
ON tblEmployee.DepartmentId = tblDepartment.Id  
**OR**  
SELECT Name, Gender, Salary, DepartmentName  
FROM tblEmployee  
LEFT JOIN tblDepartment  
ON tblEmployee.DepartmentId = tblDepartment.Id

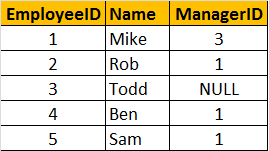
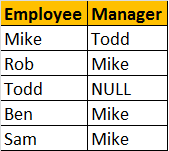
**Note:** You can use, LEFT JOIN or LEFT OUTER JOIN. OUTER keyowrd is optional  
  
**LEFT JOIN**, returns all the matching rows + non matching rows from the left table. In reality, INNER JOIN and LEFT JOIN are extensively used.  
  
**RIGHT JOIN or RIGHT OUTER JOIN**  
I want, all the rows from the right table. The query output should be, as shown below.  
  
SELECT Name, Gender, Salary, DepartmentName  
FROM tblEmployee  
RIGHT OUTER JOIN tblDepartment  
ON tblEmployee.DepartmentId = tblDepartment.Id  
**OR**  
SELECT Name, Gender, Salary, DepartmentName  
FROM tblEmployee  
RIGHT JOIN tblDepartment  
ON tblEmployee.DepartmentId = tblDepartment.Id  
  
**Note:** You can use, RIGHT JOIN or RIGHT OUTER JOIN. OUTER keyowrd is optional  
**RIGHT JOIN**, returns all the matching rows + non matching rows from the right table.  
  
**FULL JOIN or FULL OUTER JOIN**  
I want all the rows from both the tables involved in the join. The query output should be, as shown below.  
  
SELECT Name, Gender, Salary, DepartmentName  
FROM tblEmployee  
FULL OUTER JOIN tblDepartment  
ON tblEmployee.DepartmentId = tblDepartment.Id  
OR  
SELECT Name, Gender, Salary, DepartmentName  
FROM tblEmployee  
FULL JOIN tblDepartment  
ON tblEmployee.DepartmentId = tblDepartment.Id  
  
**Note:** You can use, FULLJOIN or FULL OUTER JOIN. OUTER keyowrd is optional  
**FULL JOIN**, returns all rows from both the left and right tables, including the non matching rows.  
  
**Joins Summary**  
  
  


## Self-Join

 Consider tblEmployees table shown below.  
  
Write a query which gives the following result.  
  
A MANAGER is also an EMPLOYEE. Both the, EMPLOYEE and MANAGER rows, are present in the same table. Here we are joining tblEmployee with itself using different alias names, E for Employee and M for Manager. We are using LEFT JOIN, to get the rows with ManagerId NULL. You can see in the output TODD's record is also retrieved, but the MANAGER is NULL. If you replace LEFT JOIN with INNER JOIN, you will not get TODD's record.  
Select E.Name as Employee, M.Name as Manager  
from tblEmployee E  
Left Join tblEmployee M  
On E.ManagerId = M.EmployeeId

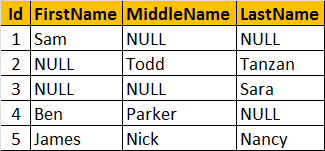
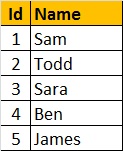
In short, joining a table with itself is called as **SELF JOIN**. SELF JOIN is not a different type of JOIN. It can be classified under any type of JOIN - INNER, OUTER or CROSS Joins. The above query is, LEFT OUTER SELF Join.

**Inner Self Join tblEmployee table:**  
Select E.Name as Employee, M.Name as Manager  
from tblEmployee E  
Inner Join tblEmployee M  
On E.ManagerId = M.EmployeeId

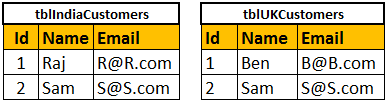
**Now Consider the Employees table below.**   
  
Writing a LEFT OUTER SELF JOIN query, which produced the following output.  
  
In the output, **MANAGER**column, for **Todd's** rows is **NULL**. I want to replace the **NULL** value, with **'No Manager'**

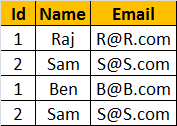
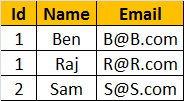
**Replacing NULL value using ISNULL() function:** We are passing 2 parameters to IsNULL() function. If M.Name returns NULL, then 'No Manager' string is used as the replacement value.  
SELECT E.Name as Employee, ISNULL(M.Name,'No Manager') as Manager  
FROM tblEmployee E  
LEFT JOIN tblEmployee M  
ON E.ManagerID = M.EmployeeID  
  
**Replacing NULL value using CASE Statement:**  
SELECT E.Name as Employee, CASE WHEN M.Name IS NULL THEN 'No Manager'   
 ELSE M.Name END as Manager  
FROM tblEmployee E  
LEFT JOIN tblEmployee M  
ON E.ManagerID = M.EmployeeID  
  
**Replacing NULL value using COALESCE() function:** COALESCE() function, returns the first NON NULL value.  
SELECT E.Name as Employee, COALESCE(M.Name, 'No Manager') as Manager  
FROM tblEmployee E  
LEFT JOIN tblEmployee M  
ON E.ManagerID = M.EmployeeID

## **Coalesce() function**

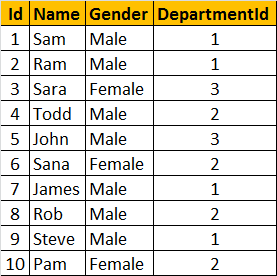
According to the MSDN Books online COALESCE() returns the first Non NULL value. Let's understand this with an example.  
Consider the Employees Table below. Not all employees have their First, Midde and Last Names filled. Some of the employees has First name missing, some of them have Middle Name missing and some of them last name.  
  
Now, let's write a query that returns the**Name of the Employee**. If an employee, has all the columns filled - **First, Middle and Last Names**, then we only want the **first name**.  
If the **FirstName is NULL**, and if **Middle and Last Names are filled** then, we only want the **middle name**. For example, Employee row with Id = 1, has the FirstName filled, so we want to retrieve his FirstName "Sam". Employee row with Id = 2, has Middle and Last names filled, but the First name is missing. Here, we want to retrieve his middle name "Todd". In short, The output of the query should be as shown below.  
  
We are passing **FirstName, MiddleName and LastName** columns as parameters to the COALESCE() function. The COALESCE() function returns the first non null value from the 3 columns.  
**SELECT Id, COALESCE(FirstName, MiddleName, LastName) AS Name**  
**FROM tblEmployee**

## Union and Union All

UNION and UNION ALL operators in SQL Server, are used to combine the result-set of two or more SELECT queries. Please consider India and UK customer tables below  


**Combining the rows of tblIndiaCustomers and tblUKCustomers using UNION ALL**  
Select Id, Name, Email from tblIndiaCustomers  
UNION ALL  
Select Id, Name, Email from tblUKCustomers  
**Query Results of UNION ALL**  
  
**Combining the rows of tblIndiaCustomers and tblUKCustomers using UNION**  
Select Id, Name, Email from tblIndiaCustomers  
UNION  
Select Id, Name, Email from tblUKCustomers  
**Query Results of UNION**  
  
**Differences between UNION and UNION ALL (Common Interview Question)**  
From the output, it is very clear that, **UNION removes duplicate** rows, where as **UNION ALL does not**. When use UNION, to remove the duplicate rows, sql server has to to do a distinct sort, which is time consuming. For this reason, UNION ALL is much faster than UNION.   
**Note:** If you want to see the cost of DISTINCT SORT, you can turn on the estimated query execution plan using CTRL + L.  
**Note:** For UNION and UNION ALL to work, the Number, Data types, and the order of the columns in the select statements should be same.  
  
**If you want to sort, the results of UNION or UNION ALL, the ORDER BY caluse should be used on the last SELECT statement as shown below.**  
Select Id, Name, Email from tblIndiaCustomers  
UNION ALL  
Select Id, Name, Email from tblUKCustomers  
UNION ALL  
Select Id, Name, Email from tblUSCustomers  
Order by Name  
  
**The following query, raises a syntax error**  
SELECT Id, Name, Email FROM tblIndiaCustomers  
ORDER BY Name  
UNION ALL  
SELECT Id, Name, Email FROM tblUKCustomers  
UNION ALL  
SELECT Id, Name, Email FROM tblUSCustomers  
  
**Difference between JOIN and UNION**  
**JOINS** and **UNIONS** are different things. However, this question is being asked very frequently now. UNION combines the result-set of two or more select queries into a single result-set which includes all the rows from all the queries in the union, where as JOINS, retrieve data from two or more tables based on logical relationships between the tables. In short, UNION combines rows from 2 or more tables, where JOINS combine columns from 2 or more table.

## Stored Procedure

A stored procedure is group of T-SQL (Transact SQL) statements. If you have a situation, where you write the same query over and over again, you can save that specific query as a stored procedure and call it just by it's name.  
  
There are several advantages of using stored procedures, which we will discuss in a later video session. In this session, we will learn how to create, execute, change and delete stored procedures.  
  
  
  
**Creating a simple stored procedure without any parameters**: This stored procedure, retrieves Name and Gender of all the employees. To create a stored procedure we use, **CREATE PROCEDURE** or **CREATE PROC** statement.  
  
Create Procedure spGetEmployees  
as  
Begin  
  Select Name, Gender from tblEmployee  
End  
  
**Note:** When naming user defined stored procedures, Microsoft recommends not to use **"sp\_"** as a prefix. All system stored procedures, are prefixed with **"sp\_"**. This avoids any ambiguity between user defined and system stored procedures and any conflicts, with some future system procedure.  
  
**To execute the stored procedure**, you can just type the procedure name and press F5, or use EXEC or EXECUTE keywords followed by the procedure name as shown below.  
1. spGetEmployees  
2. EXEC spGetEmployees  
3. Execute spGetEmployees  
  
**Note:** You can also right click on the procedure name, in object explorer in SQL Server Management Studio and select EXECUTE STORED PROCEDURE.  
  
**Creating a stored procedure with input parameters:** This SP, accepts GENDER and DEPARTMENTID parameters. Parameters and variables have an @ prefix in their name.

Create Procedure spGetEmployeesByGenderAndDepartment   
@Gender nvarchar(50),  
@DepartmentId int  
as  
Begin  
  Select Name, Gender from tblEmployee Where Gender = @Gender and DepartmentId = @DepartmentId  
End  
  
To invoke this procedure, we need to pass the value for @Gender and @DepartmentId parameters. If you don't specify the name of the parameters, you have to first pass value for @Gender parameter and then for @DepartmentId.  
EXECUTE spGetEmployeesByGenderAndDepartment 'Male', 1  
  
On the other hand, if you change the order, you will get an error stating "Error converting data type varchar to int." This is because, the value of **"Male"** is passed into @DepartmentId parameter. Since @DepartmentId is an integer, we get the type conversion error.  
**spGetEmployeesByGenderAndDepartment 1, 'Male'**  
  
When you specify the names of the parameters when executing the stored procedure the order doesn't matter.  
EXECUTE spGetEmployeesByGenderAndDepartment @DepartmentId=1, @Gender = 'Male'  
  
**To view the text, of the stored procedure**  
1. Use system stored procedure sp\_helptext 'SPName'  
OR  
2. Right Click the SP in Object explorer -> Scrip Procedure as -> Create To -> New Query Editor Window  
  
**To change the stored procedure, use ALTER PROCEDURE statement:**  
Alter Procedure spGetEmployeesByGenderAndDepartment   
@Gender nvarchar(50),  
@DepartmentId int  
as  
Begin  
  Select Name, Gender from tblEmployee Where Gender = @Gender and DepartmentId = @DepartmentId order by Name  
End  
  
**To encrypt the text of the SP**, use WITH ENCRYPTION option. Once, encrypted, you cannot view the text of the procedure, using sp\_helptext system stored procedure. There are ways to obtain the original text, which we will talk about in a later session.  
Alter Procedure spGetEmployeesByGenderAndDepartment   
@Gender nvarchar(50),  
@DepartmentId int  
WITH ENCRYPTION  
as  
Begin  
  Select Name, Gender from tblEmployee Where Gender = @Gender and DepartmentId = @DepartmentId  
End  
  
To delete the SP, use DROP PROC 'SPName' or DROP PROCEDURE 'SPName'c