/****************/
/* SQL SERVER   */
/* INTERVIEW Qs.*/
/*   INDEX   */
/****************/

--10-- JOIN 3 TABLES IN SQL SERVER
--11-- REAL TIME EXAMPLE FOR RIGHT JOIN
--12-- CAN WE JOIN TWO TABLES WITHOUT PRIMARY FOREIGN KEY RELATION
--13-- DIFFERENCE BETWEEN BLOCKING AND DEADLOCKING
--14-- SQL QUERY TO SELECT ALL NAMES THAT START WITH A GIVEN LETTER WITHOUT LIKE OPERATOR
--15-- SQL SCRIPT TO INSERT INTO MANY TO MANY TABLE

/****************/
/* SQL SERVER   */
/* INTERVIEW Qs.*/
/****************/

--10-- JOIN 3 TABLES IN SQL SERVER

In this video we will discuss joining 3 tables in SQL Server. Joining 3 tables (or even more) is very similar to joining 2 tables.

We will be using the following 3 tables in this demo.
Join 3 tables in sql server

SQL Script to create the required tables

```
Create Table Departments
(
    DepartmentID int primary key,
    DepartmentName nvarchar(50)
)
GO

Create Table Genders
(
    GenderID int primary key,
    Gender nvarchar(50)
)
GO

Create Table Employees
(
    EmployeeID int primary key,
    EmployeeName nvarchar(50),
    DepartmentID int foreign key references Departments(DepartmentID),
    GenderID int foreign key references Genders(GenderID)
```

```
)
GO

Insert into Departments values (1, 'IT')
Insert into Departments values (2, 'HR')
Insert into Departments values (3, 'Payroll')
GO

Insert into Genders values (1, 'Male')
Insert into Genders values (2, 'Female')
GO

Insert into Employees values (1, 'Mark', 1, 1)
Insert into Employees values (2, 'John', 1, 1)
Insert into Employees values (3, 'Mike', 2, 1)
Insert into Employees values (4, 'Mary', 2, 2)
Insert into Employees values (5, 'Stacy', 3, 2)
Insert into Employees values (6, 'Valarie', 3, 2)
GO
```

Write a query to join 3 the tables and retrieve EmployeeName, DepartmentName and Gender. The output should be as shown below.
joining 3 or more tables in sql server

Query:

```
SELECT EmployeeName, DepartmentName, Gender
FROM Employees
JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID
JOIN Genders ON Employees.GenderID = Genders.GenderID
```

Write a query to show the total number of employees by DEPARTMENT and by GENDER. The output should be as shown below.
group by 2 columns sql server

Query:

```
SELECT DepartmentName, Gender, COUNT(*) as TotalEmployees
FROM Employees
JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID
JOIN Genders ON Employees.GenderID = Genders.GenderID
GROUP BY DepartmentName, Gender
ORDER BY DepartmentName, Gender
```

--11-- REAL TIME EXAMPLE FOR RIGHT JOIN

In my opinion this is a very good sql server interview question. Here are the sequence of questions, one of our Youtube
channel subscribers faced in a sql server interview.

Question 1: Can you list different types of JOINS available in SQL Server
Answer: Inner Join, Left Join, Right Join, Full Join and Cross Join

Question 2: Can you tell me the purpose of Right Join?
Answer: Right Join returns all rows from the Right Table irrespective of whether a match exists in the left table or not.

Question 3: Can you give me an example?
Answer: Consider the following Departments and Employees tables.
Real time example for right join

In this case we use RIGHT JOIN To retrieve all Department and Employee names, irrespective of whether a Department has Employees or not.
real time example for right join in sql server

```
Select DepartmentName, EmployeeName
From Employees
Right Join Departments
On Employees.DepartmentID = Departments.DepartmentID
```

Question 4: I accept you have understood the purpose of Right Join.
Based on the above 2 tables, can you give me any other business case for using Right Join.

At this point the candidate being interviewed, had no other answer and he simply told the interviewer he cant think of anything else. The good news is that, inspite of not answering the last question, the candidate got this Job.

The candidate then emailed me and asked, what do you think that interviewer might be looking for here?
Here is what I think the interviewer is looking for. I may be wrong. If you can think of anything else, please feel free to leave a comment so it could help someone else.

Another business case for using RIGHT JOIN on the above 2 tables is to retrieve all the Department Names and the total number of Employees with in each department.
business case for right join

SQL Query with Right Join

```
Select DepartmentName, Count(Employees.DepartmentID) as TotalEmployees
From Employees
Right Join Departments
ON Departments.DepartmentID = Employees.DepartmentID
Group By DepartmentName
Order By TotalEmployees
```

SQL Script to create the required tables

```
Create Table Departments
(
    DepartmentID int primary key,
    DepartmentName nvarchar(50)
```

```
)
GO

Create Table Employees
(
    EmployeeID int primary key,
    EmployeeName nvarchar(50),
    DepartmentID int foreign key references Departments(DepartmentID)
)
GO

Insert into Departments values (1, 'IT')
Insert into Departments values (2, 'HR')
Insert into Departments values (3, 'Payroll')
Insert into Departments values (4, 'Admin')
GO

Insert into Employees values (1, 'Mark', 1)
Insert into Employees values (2, 'John', 1)
Insert into Employees values (3, 'Mike', 1)
Insert into Employees values (4, 'Mary', 2)
Insert into Employees values (5, 'Stacy', 2)
GO
```

--12-- CAN WE JOIN TWO TABLES WITHOUT PRIMARY FOREIGN KEY RELATION

Create Departments and Employees tables using the SQL script below

```
Create table Departments
(
    ID int not null,
    Name nvarchar(50),
    Location nvarchar(50)
)
GO

Create table Employees
(
    ID int,
    Name nvarchar(50),
    Gender nvarchar(50),
    Salary int,
    DepartmentId int
)
GO

Insert into Departments values (1, 'IT', 'New York')
Insert into Departments values (2, 'HR', 'London')
```

```
Insert into Departments values (3, 'Payroll', 'Sydney')
GO

Insert into Employees values (1, 'Mark', 'Male', 60000, 1)
Insert into Employees values (2, 'Steve', 'Male', 45000, 3)
Insert into Employees values (3, 'Ben', 'Male', 70000, 1)
Insert into Employees values (4, 'Philip', 'Male', 45000, 2)
Insert into Employees values (5, 'Mary', 'Female', 30000, 2)
Insert into Employees values (6, 'Valarie', 'Female', 35000, 3)
Insert into Employees values (7, 'John', 'Male', 80000, 1)
GO
```

Notice that ID column in Departments table is not the primary Key and DepartmentId column in Employees table is not the foreign key. But we can still join these tables using ID column from Departments table and DepartmentId column from Employees table, as both the columns involved in the join have same data type i.e int.

```
Select Employees.Name as EmployeeName, Departments.Name as DepartmentName
from Employees
join Departments on Departments.ID = Employees.DepartmentId
```

The above query produces the following output

Can we join two tables without primary foreign key relation

The obious next question is, if primary foreign key relation is not mandatory for 2 tables to be joined then what is the use of these keys?

Primary key enforces uniqueness of values over one or more columns. Since ID is not a primary key in Departments table, 2 or more departments may end up having same ID value, which makes it impossible to distinguish between them based on the ID column value.

Foreign key enforces referential integrity. Without foreign key constraint on DepartmentId column in Employees table, it is possible to insert a row into Employees table with a value for DepartmentId column that does not exist in Departments table.

The following insert statement, successfully inserts a new Employee into Employees table whose DepartmentId is 100. But we dont have a department with ID = 100 in Departments table. This means this employee row is an orphan row, and the referential integrity is lost as result

```
Insert into Employees values (8, 'Mary', 'Female', 80000, 100)
```

If we have had a foreign key constraint on DepartmentId column in Employees table, the following insert statement would have failed with the following error.

Msg 547, Level 16, State 0, Line 1
The INSERT statement conflicted with the FOREIGN KEY constraint. The conflict occurred in database "Sample", table "dbo.Departments", column 'ID'.

Lets now see how to enforces referential integrity using foreign key constraint.

Step 1 : Delete the employee record from Employees table where DepartmentId = 100
Delete from Employees where DepartmentId = 100

Step 2 : Mark ID column as primary key in Departments table

```
alter table Departments
add primary key (ID)
```

Setp 3 : Mark DepartmentId column as foreign key in Employees table

```
Alter table Employees
add foreign key(DepartmentId)
references Departments(ID)
```

Step 4 : Execute the following insert statement. Notice that an error is raised.
Insert into Employees values (8, 'Mary', 'Female', 80000, 100)

```
--Other SQL Queries used in the demo
Create Table T1
(
    ID int,
    T1Column1 nvarchar(20)
)
GO

Insert into T1 values (1, 'T1 Value 1')
Insert into T1 values (2, 'T1 Value 2')
GO

Create Table T2
(
    ID decimal,
    T2Column1 nvarchar(20)
)
GO

Insert into T2 values (1, 'T2 Value 1')
Insert into T2 values (2, 'T2 Value 2')
GO

Select T1.T1Column1, T2.T2Column1
from T1
join T2 on T1.ID = T2.ID
```

Alter Table T2 Alter Column ID nvarchar(3)

Insert into T2 values('XX', 'T2 Value 3')

--13-- DIFFERENCE BETWEEN BLOCKING AND DEADLOCKING

In this video we will discuss the difference between blocking and deadlocking.
This is one of the common SQL Server interview question.
Let us understand the difference with an example.

We will be using the following 2 tables in this demo
difference between blocking and deadlock in sql server
Difference between blocking and deadlocking

SQL Script to create the tables and populate them with test data

```
Create table TableA
(
   Id int identity primary key,
   Name nvarchar(50)
)
Go

Insert into TableA values ('Mark')
Go

Create table TableB
(
   Id int identity primary key,
   Name nvarchar(50)
)
Go

Insert into TableB values ('Mary')
Go
```

Blocking : Occurs if a transaction tries to acquire an incompatible lock
on a resource that another transaction has already locked.
The blocked transaction remains blocked until the blocking transaction
releases the lock. The following diagram explains this.

blocking in sql server

Example : Open 2 instances of SQL Server Management studio. From the first
window execute Transaction 1 code and from the second window execute Transaction
2 code. Notice that Transaction 2 is blocked by Transaction 1. Transaction 2
is allowed to move forward only when Transaction 1 completes.

```
--Transaction 1
```

```
Begin Tran
Update TableA set Name='Mark Transaction 1' where Id = 1
Waitfor Delay '00:00:10'
Commit Transaction

--Transaction 2
Begin Tran
Update TableA set Name='Mark Transaction 2' where Id = 1
Commit Transaction
```

Deadlock : Occurs when two or more transactions have a resource locked, and each transaction requests a lock on the resource that another transaction has already locked. Neither of the transactions here can move forward, as each one is waiting for the other to release the lock. So in this case, SQL Server intervenes and ends the deadlock by cancelling one of the transactions, so the other transaction can move forward. The following diagram explains this.

blocking vs deadlock in sql server

Example : Open 2 instances of SQL Server Management studio. From the first window execute Transaction 1 code and from the second window execute Transaction 2 code. Notice that there is a deadlock between Transaction 1 and Transaction 2.

```
-- Transaction 1
Begin Tran
Update TableA Set Name = 'Mark Transaction 1' where Id = 1
 -- From Transaction 2 window execute the first update statement

Update TableB Set Name = 'Mary Transaction 1' where Id = 1
 -- From Transaction 2 window execute the second update statement

Commit Transaction

-- Transaction 2
Begin Tran
Update TableB Set Name = 'Mark Transaction 2' where Id = 1
-- From Transaction 1 window execute the second update statement

Update TableA Set Name = 'Mary Transaction 2' where Id = 1
-- After a few seconds notice that one of the transactions complete
-- successfully while the other transaction is made the deadlock victim
Commit Transaction
```

--14-- SQL QUERY TO SELECT ALL NAMES THAT START WITH A GIVEN LETTER WITHOUT LIKE OPERATOR

In this video we will discuss writing a SQL query to retrieve all student names that start with letter 'M' without using the LIKE operator.

We will use the following Students table for this example
sql query without using like operator

SQL Script to create the Students table

```
Create table Students
(
    ID int primary key identity,
    Name nvarchar(50),
    Gender nvarchar(50),
    Salary int
)
Go

Insert into Students values ('Mark', 'Male', 60000)
Insert into Students values ('Steve', 'Male', 45000)
Insert into Students values ('James', 'Male', 70000)
Insert into Students values ('Mike', 'Male', 45000)
Insert into Students values ('Mary', 'Female', 30000)
Insert into Students values ('Valarie', 'Female', 35000)
Insert into Students values ('John', 'Male', 80000)
Go
```

Interview question : Write a query to select all student rows
whose Name starts with letter 'M' without using the LIKE operator

The output should be as shown below
sql search without like wildcard

If the interviewer has not mentioned not to use LIKE operator,
we would have written the query using the LIKE operator as shown below.

```
SELECT * FROM Students WHERE Name LIKE 'M%'
```

We can use any one of the following 3 SQL Server functions, to achieve exactly the same thing
CHARINDEX
LEFT
SUBSTRING

The following 3 queries retrieve all student rows whose Name starts with letter 'M'.
Notice none of the queries are using the LIKE operator.

```
SELECT * FROM Students WHERE CHARINDEX('M',Name) = 1
SELECT * FROM Students WHERE LEFT(Name, 1) = 'M'
SELECT * FROM Students WHERE SUBSTRING(Name, 1, 1) = 'M'
```

--15-- SQL SCRIPT TO INSERT INTO MANY TO MANY TABLE

In this video we will discuss how to insert data into a table that has many-to-many relationship.

We will use the following 3 tables for this example
sql many to many insert insert into junction table sql server
insert into many to many relationship

SQL Script to create the tables

```
Create table Students
(
    Id int primary key identity,
    StudentName nvarchar(50)
)
Go

Create table Courses
(
    Id int primary key identity,
    CourseName nvarchar(50)
)
Go

Create table StudentCourses
(
    StudentId int not null foreign key references Students(Id),
    CourseId int not null foreign key references Courses(Id)
)
Go
```

Students - Id column is identity column
Courses - Id column is identity column
StudentCourses - StudentId and CourseId columns are foreign keys referencing
Id column in Students and Courses tables
As you can see, StudentCourses is a bridge table that has many to many relationship
with Students and Courses tables. This means a given student can be enrolled
into many courses and a given course can have many students enrolled.

Below is the question asked in an interview for SQL Server Developer role.

Write a SQL script to insert data into StudentCourses table.
Here are the rules that your script should follow.

1. There will be 2 inputs for the script
Student Name - The name of the student who wants to enrol into a course
Course Name - The name of the course the student wants to enrol into

2. If the student is already in the Students table, then use that existing
Student Id. If the student is not already in the Students table, then a row

for that student must be inserted into the Students table, and use that new student id.

3. Along the same lines, if the course is already in the Courses table, then
use that existing Course Id. If the course is not already in the Courses table,
then a row for that course must be inserted into the Courses table, and use that new course id.

4. There should be no duplicate student course enrolments, i.e a given student must not
be enrolled in the same course twice. For example, Tom must not be enrolled
in C# course twice.

Answer : To avoid duplicate student course enrolments create a composite
primary key on StudentId and CourseId columns in StudentCourses table.
With this composite primary key in place, if someone tries to enroll the
same student in the same course again we get violation of primary key constraint error.

```
Alter table StudentCourses
Add Constraint PK_StudentCourses
Primary Key Clustered (CourseId, StudentId)
```

Here is the SQL script that inserts data into the 3 tables as expected

```
Declare @StudentName nvarchar(50) = 'Sam'
Declare @CourseName nvarchar(50) = 'SQL Server'

Declare @StudentId int
Declare @CourseId int

-- If the student already exists, use the existing student ID
Select @StudentId = Id
from Students
where StudentName = @StudentName
-- If the course already exists, use the existing course ID
Select @CourseId = Id
from Courses
where CourseName = @CourseName

-- If the student does not exist in the Students table
If (@StudentId is null)
Begin
    -- Insert the student
    Insert into Students values(@StudentName)
    -- Get the Id of the student
    Select @StudentId = SCOPE_IDENTITY()
End

-- If the course does not exist in the Courses table
If (@CourseId is null)
Begin
    -- Insert the course
```

```
    Insert into Courses values(@CourseName)
    -- Get the Id of the course
    Select @CourseId = SCOPE_IDENTITY()
End

-- Insert StudentId & CourseId in StudentCourses table

Insert into StudentCourses
values(@StudentId, @CourseId)
```

If required, we can very easily convert this into a stored procedure as shown below.

```
Create procedure spInsertIntoStudentCourses
@StudentName nvarchar(50),
@CourseName nvarchar(50)
as
Begin

    Declare @StudentId int
    Declare @CourseId int

    Select @StudentId = Id
    from Students
    where StudentName = @StudentName

    Select @CourseId = Id
    from Courses
    where CourseName = @CourseName

    If (@StudentId is null)
    Begin
        Insert into Students values(@StudentName)
        Select @StudentId = SCOPE_IDENTITY()
    End

    If (@CourseId is null)
    Begin
        Insert into Courses values(@CourseName)
        Select @CourseId = SCOPE_IDENTITY()
    End

    Insert into StudentCourses values(@StudentId, @CourseId)

End

Use the following statement to execute the stored procedure
Execute spInsertIntoStudentCourses 'Tom','C#'/****************/
/* SQL SERVER   */
```

/* INTERVIEW Qs.*/
/*   INDEX   */
/****************/

--10-- JOIN 3 TABLES IN SQL SERVER
--11-- REAL TIME EXAMPLE FOR RIGHT JOIN
--12-- CAN WE JOIN TWO TABLES WITHOUT PRIMARY FOREIGN KEY RELATION
--13-- DIFFERENCE BETWEEN BLOCKING AND DEADLOCKING
--14-- SQL QUERY TO SELECT AALL NAMES THAT START WITH A GIVEN LETTER WITHOUT LIKE OPERATOR
--15-- SQL SCRIPT TO INSERT INTO MANY TO MANY TABLE

/****************/
/* SQL SERVER   */
/* INTERVIEW Qs.*/
/****************/

--10-- JOIN 3 TABLES IN SQL SERVER

In this video we will discuss joining 3 tables in SQL Server. Joining 3 tables (or even more) is very similar to joining 2 tables.

We will be using the following 3 tables in this demo.
Join 3 tables in sql server

SQL Script to create the required tables

```
Create Table Departments
(
    DepartmentID int primary key,
    DepartmentName nvarchar(50)
)
GO

Create Table Genders
(
    GenderID int primary key,
    Gender nvarchar(50)
)
GO

Create Table Employees
(
    EmployeeID int primary key,
    EmployeeName nvarchar(50),
    DepartmentID int foreign key references Departments(DepartmentID),
    GenderID int foreign key references Genders(GenderID)
)
GO
```

```
Insert into Departments values (1, 'IT')
Insert into Departments values (2, 'HR')
Insert into Departments values (3, 'Payroll')
GO

Insert into Genders values (1, 'Male')
Insert into Genders values (2, 'Female')
GO

Insert into Employees values (1, 'Mark', 1, 1)
Insert into Employees values (2, 'John', 1, 1)
Insert into Employees values (3, 'Mike', 2, 1)
Insert into Employees values (4, 'Mary', 2, 2)
Insert into Employees values (5, 'Stacy', 3, 2)
Insert into Employees values (6, 'Valarie', 3, 2)
GO
```

Write a query to join 3 the tables and retrieve EmployeeName, DepartmentName and Gender. The output should be as shown below.
joining 3 or more tables in sql server

Query:

```
SELECT EmployeeName, DepartmentName, Gender
FROM Employees
JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID
JOIN Genders ON Employees.GenderID = Genders.GenderID
```

Write a query to show the total number of employees by DEPARTMENT and by GENDER. The output should be as shown below.
group by 2 columns sql server

Query:

```
SELECT DepartmentName, Gender, COUNT(*) as TotalEmployees
FROM Employees
JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID
JOIN Genders ON Employees.GenderID = Genders.GenderID
GROUP BY DepartmentName, Gender
ORDER BY DepartmentName, Gender
```

--11-- REAL TIME EXAMPLE FOR RIGHT JOIN

In my opinion this is a very good sql server interview question. Here are the sequence of questions, one of our Youtube
channel subscribers faced in a sql server interview.

Question 1: Can you list different types of JOINS available in SQL Server
Answer: Inner Join, Left Join, Right Join, Full Join and Cross Join

Question 2: Can you tell me the purpose of Right Join?
Answer: Right Join returns all rows from the Right Table irrespective of whether a match exists in the left table or not.

Question 3: Can you give me an example?
Answer: Consider the following Departments and Employees tables.
Real time example for right join

In this case we use RIGHT JOIN To retrieve all Department and Employee names, irrespective of whether a Department has Employees or not.
real time example for right join in sql server

```
Select DepartmentName, EmployeeName
From Employees
Right Join Departments
On Employees.DepartmentID = Departments.DepartmentID
```

Question 4: I accept you have understood the purpose of Right Join.
Based on the above 2 tables, can you give me any other business case for using Right Join.

At this point the candidate being interviewed, had no other answer and he simply told the interviewer he cant think of anything else. The good news is that, inspite of not answering the last question, the candidate got this Job.

The candidate then emailed me and asked, what do you think that interviewer might be looking for here?
Here is what I think the interviewer is looking for. I may be wrong. If you can think of anything else, please feel free to leave a comment so it could help someone else.

Another business case for using RIGHT JOIN on the above 2 tables is to retrieve all the Department Names and the total number of Employees with in each department.
business case for right join

SQL Query with Right Join

```
Select DepartmentName, Count(Employees.DepartmentID) as TotalEmployees
From Employees
Right Join Departments
ON Departments.DepartmentID = Employees.DepartmentID
Group By DepartmentName
Order By TotalEmployees
```

SQL Script to create the required tables

```
Create Table Departments
(
    DepartmentID int primary key,
    DepartmentName nvarchar(50)
)
GO
```

```
Create Table Employees
(
    EmployeeID int primary key,
    EmployeeName nvarchar(50),
    DepartmentID int foreign key references Departments(DepartmentID)
)
GO

Insert into Departments values (1, 'IT')
Insert into Departments values (2, 'HR')
Insert into Departments values (3, 'Payroll')
Insert into Departments values (4, 'Admin')
GO

Insert into Employees values (1, 'Mark', 1)
Insert into Employees values (2, 'John', 1)
Insert into Employees values (3, 'Mike', 1)
Insert into Employees values (4, 'Mary', 2)
Insert into Employees values (5, 'Stacy', 2)
GO
```

--12-- CAN WE JOIN TWO TABLES WITHOUT PRIMARY FOREIGN KEY RELATION

Create Departments and Employees tables using the SQL script below

```
Create table Departments
(
    ID int not null,
    Name nvarchar(50),
    Location nvarchar(50)
)
GO

Create table Employees
(
    ID int,
    Name nvarchar(50),
    Gender nvarchar(50),
    Salary int,
    DepartmentId int
)
GO

Insert into Departments values (1, 'IT', 'New York')
Insert into Departments values (2, 'HR', 'London')
Insert into Departments values (3, 'Payroll', 'Sydney')
GO
```

```
Insert into Employees values (1, 'Mark', 'Male', 60000, 1)
Insert into Employees values (2, 'Steve', 'Male', 45000, 3)
Insert into Employees values (3, 'Ben', 'Male', 70000, 1)
Insert into Employees values (4, 'Philip', 'Male', 45000, 2)
Insert into Employees values (5, 'Mary', 'Female', 30000, 2)
Insert into Employees values (6, 'Valarie', 'Female', 35000, 3)
Insert into Employees values (7, 'John', 'Male', 80000, 1)
GO
```

Notice that ID column in Departments table is not the primary Key and
DepartmentId column in Employees table is not the foreign key.
But we can still join these tables using ID column from Departments table
and DepartmentId column from Employees table, as both the columns involved
in the join have same data type i.e int.

```
Select Employees.Name as EmployeeName, Departments.Name as DepartmentName
from Employees
join Departments on Departments.ID = Employees.DepartmentId
```

The above query produces the following output

Can we join two tables without primary foreign key relation

The obious next question is, if primary foreign key relation is
not mandatory for 2 tables to be joined then what is the use of these keys?

Primary key enforces uniqueness of values over one or more columns.
Since ID is not a primary key in Departments table, 2 or more departments may
end up having same ID value, which makes it impossible to distinguish between
them based on the ID column value.

Foreign key enforces referential integrity. Without foreign key constraint on
DepartmentId column in Employees table, it is possible to insert a row into
Employees table with a value for DepartmentId column that does not exist in
Departments table.

The following insert statement, successfully inserts a new Employee into
Employees table whose DepartmentId is 100. But we dont have a department
with ID = 100 in Departments table. This means this employee row is an
orphan row, and the referential integrity is lost as result

```
Insert into Employees values (8, 'Mary', 'Female', 80000, 100)
```

If we have had a foreign key constraint on DepartmentId column in Employees table,
the following insert statement would have failed with the following error.

```
Msg 547, Level 16, State 0, Line 1
The INSERT statement conflicted with the FOREIGN KEY constraint. The conflict occurred
```

in database "Sample", table "dbo.Departments", column 'ID'.

Lets now see how to enforces referential integrity using foreign key constraint.

Step 1 : Delete the employee record from Employees table where DepartmentId = 100
Delete from Employees where DepartmentId = 100

Step 2 : Mark ID column as primary key in Departments table

```
alter table Departments
add primary key (ID)
```

Setp 3 : Mark DepartmentId column as foreign key in Employees table

```
Alter table Employees
add foreign key(DepartmentId)
references Departments(ID)
```

Step 4 : Execute the following insert statement. Notice that an error is raised.
Insert into Employees values (8, 'Mary', 'Female', 80000, 100)

```
--Other SQL Queries used in the demo
Create Table T1
(
    ID int,
    T1Column1 nvarchar(20)
)
GO

Insert into T1 values (1, 'T1 Value 1')
Insert into T1 values (2, 'T1 Value 2')
GO

Create Table T2
(
    ID decimal,
    T2Column1 nvarchar(20)
)
GO

Insert into T2 values (1, 'T2 Value 1')
Insert into T2 values (2, 'T2 Value 2')
GO

Select T1.T1Column1, T2.T2Column1
from T1
join T2 on T1.ID = T2.ID

Alter Table T2 Alter Column ID nvarchar(3)
```

Insert into T2 values('XX', 'T2 Value 3')

--13-- DIFFERENCE BETWEEN BLOCKING AND DEADLOCKING

In this video we will discuss the difference between blocking and deadlocking.
This is one of the common SQL Server interview question.
Let us understand the difference with an example.

We will be using the following 2 tables in this demo
difference between blocking and deadlock in sql server
Difference between blocking and deadlocking

SQL Script to create the tables and populate them with test data

```
Create table TableA
(
   Id int identity primary key,
   Name nvarchar(50)
)
Go

Insert into TableA values ('Mark')
Go

Create table TableB
(
   Id int identity primary key,
   Name nvarchar(50)
)
Go

Insert into TableB values ('Mary')
Go
```

Blocking : Occurs if a transaction tries to acquire an incompatible lock
on a resource that another transaction has already locked.
The blocked transaction remains blocked until the blocking transaction
releases the lock. The following diagram explains this.

blocking in sql server

Example : Open 2 instances of SQL Server Management studio. From the first
window execute Transaction 1 code and from the second window execute Transaction
2 code. Notice that Transaction 2 is blocked by Transaction 1. Transaction 2
is allowed to move forward only when Transaction 1 completes.

```
--Transaction 1
Begin Tran
Update TableA set Name='Mark Transaction 1' where Id = 1
```

```
Waitfor Delay '00:00:10'
Commit Transaction

--Transaction 2
Begin Tran
Update TableA set Name='Mark Transaction 2' where Id = 1
Commit Transaction
```

Deadlock : Occurs when two or more transactions have a resource locked, and each transaction requests a lock on the resource that another transaction has already locked. Neither of the transactions here can move forward, as each one is waiting for the other to release the lock. So in this case, SQL Server intervenes and ends the deadlock by cancelling one of the transactions, so the other transaction can move forward. The following diagram explains this.

blocking vs deadlock in sql server

Example : Open 2 instances of SQL Server Management studio. From the first window execute Transaction 1 code and from the second window execute Transaction 2 code. Notice that there is a deadlock between Transaction 1 and Transaction 2.

```
-- Transaction 1
Begin Tran
Update TableA Set Name = 'Mark Transaction 1' where Id = 1
 -- From Transaction 2 window execute the first update statement

Update TableB Set Name = 'Mary Transaction 1' where Id = 1
 -- From Transaction 2 window execute the second update statement

Commit Transaction

-- Transaction 2
Begin Tran
Update TableB Set Name = 'Mark Transaction 2' where Id = 1
-- From Transaction 1 window execute the second update statement

Update TableA Set Name = 'Mary Transaction 2' where Id = 1
-- After a few seconds notice that one of the transactions complete
-- successfully while the other transaction is made the deadlock victim
Commit Transaction
```

--14-- SQL QUERY TO SELECT ALL NAMES THAT START WITH A GIVEN LETTER WITHOUT LIKE OPERATOR

In this video we will discuss writing a SQL query to retrieve all student names that start with letter 'M' without using the LIKE operator.

We will use the following Students table for this example
sql query without using like operator

SQL Script to create the Students table

```
Create table Students
(
    ID int primary key identity,
    Name nvarchar(50),
    Gender nvarchar(50),
    Salary int
)
Go

Insert into Students values ('Mark', 'Male', 60000)
Insert into Students values ('Steve', 'Male', 45000)
Insert into Students values ('James', 'Male', 70000)
Insert into Students values ('Mike', 'Male', 45000)
Insert into Students values ('Mary', 'Female', 30000)
Insert into Students values ('Valarie', 'Female', 35000)
Insert into Students values ('John', 'Male', 80000)
Go
```

Interview question : Write a query to select all student rows
whose Name starts with letter 'M' without using the LIKE operator

The output should be as shown below
sql search without like wildcard

If the interviewer has not mentioned not to use LIKE operator,
we would have written the query using the LIKE operator as shown below.

```
SELECT * FROM Students WHERE Name LIKE 'M%'
```

We can use any one of the following 3 SQL Server functions, to achieve exactly the same thing
CHARINDEX
LEFT
SUBSTRING

The following 3 queries retrieve all student rows whose Name starts with letter 'M'.
Notice none of the queries are using the LIKE operator.

```
SELECT * FROM Students WHERE CHARINDEX('M',Name) = 1
SELECT * FROM Students WHERE LEFT(Name, 1) = 'M'
SELECT * FROM Students WHERE SUBSTRING(Name, 1, 1) = 'M'
```

--15-- SQL SCRIPT TO INSERT INTO MANY TO MANY TABLE

In this video we will discuss how to insert data into a table that has many-to-many relationship.

We will use the following 3 tables for this example
sql many to many insert insert into junction table sql server
insert into many to many relationship

SQL Script to create the tables

```
Create table Students
(
    Id int primary key identity,
    StudentName nvarchar(50)
)
Go

Create table Courses
(
    Id int primary key identity,
    CourseName nvarchar(50)
)
Go

Create table StudentCourses
(
    StudentId int not null foreign key references Students(Id),
    CourseId int not null foreign key references Courses(Id)
)
Go
```

Students - Id column is identity column
Courses - Id column is identity column
StudentCourses - StudentId and CourseId columns are foreign keys referencing
Id column in Students and Courses tables
As you can see, StudentCourses is a bridge table that has many to many relationship
with Students and Courses tables. This means a given student can be enrolled
into many courses and a given course can have many students enrolled.

Below is the question asked in an interview for SQL Server Developer role.

Write a SQL script to insert data into StudentCourses table.
Here are the rules that your script should follow.

1. There will be 2 inputs for the script
Student Name - The name of the student who wants to enrol into a course
Course Name - The name of the course the student wants to enrol into

2. If the student is already in the Students table, then use that existing
Student Id. If the student is not already in the Students table, then a row
for that student must be inserted into the Students table, and use that new student id.

3. Along the same lines, if the course is already in the Courses table, then
use that existing Course Id. If the course is not already in the Courses table,
then a row for that course must be inserted into the Courses table, and use that new course id.

4. There should be no duplicate student course enrolments, i.e a given student must not
be enrolled in the same course twice. For example, Tom must not be enrolled
in C# course twice.

Answer : To avoid duplicate student course enrolments create a composite
primary key on StudentId and CourseId columns in StudentCourses table.
With this composite primary key in place, if someone tries to enroll the
same student in the same course again we get violation of primary key constraint error.

```
Alter table StudentCourses
Add Constraint PK_StudentCourses
Primary Key Clustered (CourseId, StudentId)
```

Here is the SQL script that inserts data into the 3 tables as expected

```
Declare @StudentName nvarchar(50) = 'Sam'
Declare @CourseName nvarchar(50) = 'SQL Server'

Declare @StudentId int
Declare @CourseId int

-- If the student already exists, use the existing student ID
Select @StudentId = Id
from Students
where StudentName = @StudentName
-- If the course already exists, use the existing course ID
Select @CourseId = Id
from Courses
where CourseName = @CourseName

-- If the student does not exist in the Students table
If (@StudentId is null)
Begin
    -- Insert the student
    Insert into Students values(@StudentName)
    -- Get the Id of the student
    Select @StudentId = SCOPE_IDENTITY()
End

-- If the course does not exist in the Courses table
If (@CourseId is null)
Begin
    -- Insert the course
    Insert into Courses values(@CourseName)
    -- Get the Id of the course
```

```
    Select @CourseId = SCOPE_IDENTITY()
End

-- Insert StudentId & CourseId in StudentCourses table

Insert into StudentCourses
values(@StudentId, @CourseId)
```

If required, we can very easily convert this into a
stored procedure as shown below.

```
Create procedure spInsertIntoStudentCourses
@StudentName nvarchar(50),
@CourseName nvarchar(50)
as
Begin

    Declare @StudentId int
    Declare @CourseId int

    Select @StudentId = Id
    from Students
    where StudentName = @StudentName

    Select @CourseId = Id
    from Courses
    where CourseName = @CourseName

    If (@StudentId is null)
    Begin
        Insert into Students values(@StudentName)
        Select @StudentId = SCOPE_IDENTITY()
    End

    If (@CourseId is null)
    Begin
        Insert into Courses values(@CourseName)
        Select @CourseId = SCOPE_IDENTITY()
    End

    Insert into StudentCourses values(@StudentId, @CourseId)

End
```

Use the following statement to execute the stored procedure

```
Execute spInsertIntoStudentCourses 'Tom','C#'
```