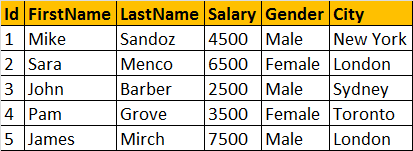
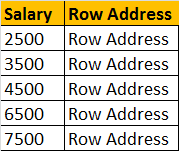
**Suggested SQL Server Videos before watching this video**  
[Part 35 - Index basics](http://csharp-video-tutorials.blogspot.com/2012/09/indexes-in-sql-server-part-35.html)  
[Part 36 - Clustered and Nonclustered indexes](http://csharp-video-tutorials.blogspot.com/2012/09/clustered-and-non-clustered-indexes.html)  
[Part 37 - Unique and Non-Unique Indexes](http://csharp-video-tutorials.blogspot.com/2012/09/unique-and-non-unique-indexes-part-37.html)  
  
**In this video session, we talk about the advantages and disadvantages of indexes**. We wil also talk about a concept called **covering queries**.   
  
   
  
   
  
   
  
**In Part 35, we have learnt that**, Indexes are used by queries to find data quickly. In this part, we will learn about the different queries that can benefit from indexes.  
  
**Create Employees table**  
CREATE TABLE [tblEmployee]  
(  
 [Id] int Primary Key,  
 [FirstName] nvarchar(50),  
 [LastName] nvarchar(50),  
 [Salary] int,  
 [Gender] nvarchar(10),  
 [City] nvarchar(50)  
)  
  
**Insert sample data:**  
Insert into tblEmployee Values(1,'Mike', 'Sandoz',4500,'Male','New York')  
Insert into tblEmployee Values(2,'Sara', 'Menco',6500,'Female','London')

Insert into tblEmployee Values(3,'John', 'Barber',2500,'Male','Sydney')  
Insert into tblEmployee Values(4,'Pam', 'Grove',3500,'Female','Toronto')  
Insert into tblEmployee Values(5,'James', 'Mirch',7500,'Male','London')  
  
**Create a Non-Clustered Index on Salary Column**  
Create NonClustered Index IX\_tblEmployee\_Salary  
On tblEmployee (Salary Asc)  
  
**Data from tblEmployee table**  
  
  
**NonClustered Index**  
  
  
  
**The following select query benefits from the index on the Salary column**, because the salaries are sorted in ascending order in the index. From the index, it's easy to identify the records where salary is between 4000 and 8000, and using the row address the corresponding records from the table can be fetched quickly.  
Select \* from tblEmployee where Salary > 4000 and Salary < 8000  
  
**Not only, the SELECT statement, even the following DELETE and UPDATE**statements can also benefit from the index. To update or delete a row, SQL server needs to first find that row, and the index can help in searching and finding that specific row quickly.  
Delete from tblEmployee where Salary = 2500  
Update tblEmployee Set Salary = 9000 where Salary = 7500  
  
**Indexes can also help queries**, that ask for sorted results. Since the Salaries are already sorted, the database engine, simply scans the index from the first entry to the last entry and retrieve the rows in sorted order. This avoids, sorting of rows during query execution, which can significantly imrpove the processing time.  
Select \* from tblEmployee order by Salary  
  
**The index on the Salary column**, can also help the query below, by scanning the index in reverse order.  
Select \* from tblEmployee order by Salary Desc  
  
**GROUP BY queries can also benefit from indexes**. To group the Employees with the same salary, the query engine, can use the index on Salary column, to retrieve the already sorted salaries. Since matching salaries are present in consecutive index entries, it is to count the total number of Employees  at each Salary quickly.   
Select Salary, COUNT(Salary) as Total  
from tblEmployee  
Group By Salary  
  
**Diadvantages of Indexes:**  
**Additional Disk Space**: Clustered Index does not, require any additional storage. Every Non-Clustered index requires additional space as it is stored separately from the table.The amount of space required will depend on the size of the table, and the number and types of columns used in the index.  
  
**Insert Update and Delete statements can become slow**: When **DML** (Data Manipulation Language) statements (**INSERT, UPDATE, DELETE**) modifies data in a table, the data in all the indexes also needs to be updated. Indexes can help, to search and locate the rows, that we want to delete, but too many indexes to update can actually hurt the performance of data modifications.  
  
**What is a covering query?**  
**If all the columns** that you have requested in the SELECT clause of query, are present in the index, then there is no need to lookup in the table again. The requested columns data can simply be returned from the index.  
  
**A clustered index**, always covers a query, since it contains all of the data in a table. A composite index is an index on two or more columns. Both clustered and nonclustered indexes can be composite indexes. To a certain extent, a composite index, can cover a query.