SQL Concepts

# Basics



## SQL Commands

https://www.tutorialgateway.org/get-database-names-from-sql-server/

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| DDL | DML | TCL | DCL |  |  |
| Create | Select | Commit | Grant |  |  |
| Alter | Insert | Roll back | Revoke |  |  |
| Drop | update | Save point |  |  |  |
| Truncate | delete | Set transaction |  |  |  |
| Rename |  |  |  |  |  |

## SQL Command Execution Order

From 🡪 Join🡪where🡪group by🡪having🡪select🡪Order by🡪top🡪aggregateFunc

## Keys

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Candidate key | Primary Key | Unique key | Super Key | Foreign Key |
| Identify other column in table | Unique  +can’t be null | Can have null | Candidate key  + non-prime attribute | **Anomalies** insertion  deletion  Updation   on base and referencing table |

## Normalization

Remove duplicity  
**avoid anomalies**1. Insert  
2. Update  
3. delete

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1st NF | 2nd NF | 3rd NF | BCNF | 4th NF | 5th NF |
| No column will have Multivalued attribute | No Partial dependency b/w prime and nonprime attribute | No Transitive dependency NonPrime should not determine Non prime | Non prime should not determine anything | No Multivalued dependency | It should be loss less join, for this use candidate key to divide |
|  |  | LHS- superKey RHS- prime, Not we |  |  |  |
|  |  | LHS-Non prime RHS-Prime |  |  |  |

## Query Optimization

https://www.databasejournal.com/features/mssql/article.php/1565961/SQL-Server-Stored-Procedures-Optimization-Tips.htm

* Set No Count On-
  + this mean the information of “count of how many records updated” it will not be returned. This gets returned in when each statement is executed
* Use with operator to avoid recompilation
  + It cached the query output which we can reuse multiple time
* Make SP such that we can reuse the execution plan
  + Don’t use DDL
  + Don’t use temp table
* Cursor- replace it with Join
* Co-related queries should be replaced with Join
* If you have a very large stored procedure, try to break down the stored procedure into several sub-procedures, and call them from a controlling stored procedure.
* Use the sp\_executesql stored procedure instead of the EXECUTE statement
  + The sp\_executesql stored procedure supports parameters. So, using the sp\_executesql stored procedure instead of the EXECUTE statement improves readability of your code when many parameters are used.
* Don’t prefix procedure with sp

## Window Function

<https://www.sqlshack.com/use-window-functions-sql-server/>

https://thinketl.com/sql-analytic-functions-interview-questions/

https://towardsdatascience.com/sql-window-function-demonstrated-with-real-interview-questions-from-leetcode-e83e28edaabc

* Aggregate Window Functions  
  SUM(), MAX(), MIN(), AVG(). COUNT()
* Ranking Window Functions  
  RANK(), DENSE\_RANK(), ROW\_NUMBER(), NTILE()
* Value Window Functions  
  LAG(), LEAD(), FIRST\_VALUE(), LAST\_VALUE()

|  |  |
| --- | --- |
| **3**  **4**  **5** | **SELECT order\_id, order\_date, customer\_name, city, order\_amount**  **,SUM(order\_amount) OVER(PARTITION BY city) as grand\_total**  **FROM [dbo].[Orders]**  SELECT dept\_id, last\_name, salary,  LEAD (salary,1) OVER (PARTITION BY dept\_id ORDER BY salary) AS next\_highest\_salary  FROM employees;  SELECT Employee\_Id, First\_Name, Department\_Id, Hire\_date, LAG(Hire\_date) OVER(PARTITION BY Department\_Id ORDER BY Hire\_date) as PREV\_HIREDATE FROM EMPLOYEES  For employee 102, the PREV\_HIREDATE is NULL as he is the oldest joinee in the department. |

CREATE TABLE [dbo].[Orders]

(

order\_id INT,

order\_date DATE,

customer\_name VARCHAR(250),

city VARCHAR(100),

order\_amount MONEY

)

INSERT INTO [dbo].[Orders]

SELECT '1001','04/01/2017','David Smith','GuildFord',10000

UNION ALL

SELECT '1002','04/02/2017','David Jones','Arlington',20000

UNION ALL

SELECT '1003','04/03/2017','John Smith','Shalford',5000

UNION ALL

SELECT '1004','04/04/2017','Michael Smith','GuildFord',15000

UNION ALL

SELECT '1005','04/05/2017','David Williams','Shalford',7000

UNION ALL

SELECT '1006','04/06/2017','Paum Smith','GuildFord',25000

UNION ALL

SELECT '1007','04/10/2017','Andrew Smith','Arlington',15000

UNION ALL

SELECT '1008','04/11/2017','David Brown','Arlington',2000

UNION ALL

SELECT '1009','04/20/2017','Robert Smith','Shalford',1000

UNION ALL

SELECT '1010','04/25/2017','Peter Smith','GuildFord',500

## Grouping set

<https://www.sqlservertutorial.net/sql-server-basics/sql-server-cube/>

revenue based on department, product for a year

select department , product , avg(sale)

from company

group by grouping sets( department , product),(department),(product), ()

## Cube

1. select department , product , avg(sale)
2. from company
3. group by cube( department , product)

this will return 2^n dimension

find only department wise

having (department as not null & product is null)

find only product wise

having (product as not null)

find both

## Pivot

CREATE TABLE Grades(

[Student] VARCHAR(50),

[Subject] VARCHAR(50),

[Marks] INT

)

GO

INSERT INTO Grades VALUES

('Jacob','Mathematics',100),

('Jacob','Science',95),

('Jacob','Geography',90),

('Amilee','Mathematics',90),

('Amilee','Science',90),

('Amilee','Geography',100)

GO

select \*

from

(select [Student],[Subject],[Marks]

from [dbo].[Grades]

)a

PIVOT (

SUM([Marks])

FOR [Subject]

IN (

[Mathematics],

[Science],

[Geography]

)

) AS PivotTable

CREATE PROCEDURE dbo.DynamicPivotTableInSql

  @ColumnToPivot  NVARCHAR(255),

  @ListToPivot    NVARCHAR(255)

AS

BEGIN

  DECLARE @SqlStatement NVARCHAR(MAX)

  SET @SqlStatement = N'

    SELECT \* FROM (

      SELECT

        [Student],

        [Subject],

        [Marks]

      FROM Grades

    ) StudentResults

    PIVOT (

      SUM([Marks])

      FOR ['+@ColumnToPivot+']

      IN (

        '+@ListToPivot+'

      )

    ) AS PivotTable

  ';

  EXEC(@SqlStatement)

END

1. Analytic Function

# With Clause

 The SQL WITH clause allows you to give a sub-query block a name

We avoid writing same block of code every time

WITH temporaryTable(averageValue) as

(SELECT avg(Salary)

from Employee),

SELECT EmployeeID,Name, Salary

FROM Employee, temporaryTable

WHERE Employee.Salary > temporaryTable.averageValue;

# Over Clause

1. [Ranking functions](https://docs.microsoft.com/en-us/sql/t-sql/functions/ranking-functions-transact-sql?view=sql-server-ver15)
2. [Aggregate functions](https://docs.microsoft.com/en-us/sql/t-sql/functions/aggregate-functions-transact-sql?view=sql-server-ver15)
3. [Analytic functions](https://docs.microsoft.com/en-us/sql/t-sql/functions/analytic-functions-transact-sql?view=sql-server-ver15)
4. [NEXT VALUE FOR function](https://docs.microsoft.com/en-us/sql/t-sql/functions/next-value-for-transact-sql?view=sql-server-ver15)

# SQL Analyic function

# CTE – Common table Expression

Refer - https://www.sqlshack.com/sql-server-common-table-expressions-cte/

A Common Table Expression, also called as CTE in short form, is a temporary named result set that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement. The CTE can also be used in a View.

ROWNUMBER

Rank,-------two repetivitve value will skip

Dense Rank---- repetive will include  
  
Union will give common records  
union all will check complete row   
  
  
dense rank , yusing partition by

# https://learnsql.com/blog/difference-between-count-distinct/#:~:text=As%20you've%20already%20learned,column%20while%20excluding%20NULL%20values.&text=Always%20remember%3A%20COUNT(column%20name,given%20column%20is%20NOT%20NULL.

# Snow flake schema, star schema

# How to implement Linear regression and logistic regression in random forest

Azure

Hot storage /cold storage

Data lake

Data factory

Data warehouse

# SQL Functions.

## Count

* COUNT([ALL | DISTINCT ] expression)
* COUNT(\*)
* COUNT(\*) counts the number of items in a set. It includes NULL, non null and duplicate values
* COUNT(ALL expression) evaluates the expression for each row in a set and returns the number of non-null values.
* COUNT(DISTINCT expression) evaluates the expression for each row in a set, and returns the number of unique, non-null values.

## Pattern Matching

* Like
* Ilike (case insensitive)
* %
* \_

## Shortcut

* we can give order of attribute from select statement in order by column.
* If two table has same column name , we can use “using()” instead of on clause.

# SQL Basic

## Why do we define Schema?

Lets say we have a company database, in this we have many tables, from sales, HR, Distribution , then to differentiate the table we define schema.

Schema uniquely identifies similar tables.

## Sorting Data

### Order by

* Sort a result set by one column in ascending order
* Sort a result set by one column in descending order
* Sort a result set by multiple columns
* Sort a result set by multiple columns and different orders
* Sort a result set by a column that is not in the select list
* Sort a result set by an expression
* Sort by ordinal positions of columns

SELECT

first\_name,

last\_name

FROM

sales.customers

ORDER BY

1,

2;

## Limiting Rows

### SQL Offset fetch

ORDER BY column\_list [ASC |DESC]

OFFSET offset\_row\_count {ROW | ROWS}

FETCH {FIRST | NEXT} fetch\_row\_count {ROW | ROWS} ONLY

* The OFFSET clause specifies the number of rows to skip before starting to return rows from the query
* The FETCH clause specifies the number of rows to return after the OFFSET clause has been processed.
* The OFFSET clause is mandatory while the FETCH clause is optional.
* Note that you must use the OFFSET and FETCH clauses with the ORDER BY clause. Otherwise, you will get an error.
* The OFFSET and FETCH clauses are preferable for implementing the query paging solution than the TOP clause.

### Top

SELECT TOP (expression) [PERCENT]

[WITH TIES]

FROM

table\_name

ORDER BY

column\_name;

* **PERCENT**

The PERCENT keyword indicates that the query returns the first N percentage of rows, where N is the result of the expression.

* **WITH TIES**

The WITH TIES allows you to return more rows with values that match the last row in the limited result set. Note that WITH TIES may cause more rows to be returned than you specify in the expression.

For example, if you want to return the most expensive products, you can use the TOP 1. However, if two or more products have the same prices as the most expensive product, then you miss the other most expensive products in the result set.

To avoid this, you can use TOP 1 WITH TIES. It will include not only the first expensive product but also the second one, and so on.

## Filtering Data

### Distinct

SELECT DISTINCT

column\_name1,

column\_name2 ,

...

FROM

table\_name;SELECT DISTINCT

column\_name

FROM

table\_name;

* The query uses the combination of values in all specified columns in the SELECT list to evaluate the uniqueness.
* If you apply the DISTINCT clause to a column that has NULL, the DISTINCT clause will keep only one NULL and eliminates the other. In other words, the DISTINCT clause treats all NULL “values” as the same value.

### DISTINCT vs. GROUP BY

* Both DISTINCT and GROUP BY clause reduces the number of returned rows in the result set by removing the duplicates.
* However, you should use the GROUP BY clause when you want to apply an [aggregate function](https://www.sqlservertutorial.net/sql-server-aggregate-functions/) on one or more columns.

### Where

SELECT

select\_list

FROM

table\_name

WHERE

search\_condition;

* In the WHERE clause, you specify a search condition to filter rows returned by the FROM clause. The WHERE clause only returns the rows that cause the search condition to evaluate to TRUE.
* The search condition is a logical expression or a combination of multiple logical expressions.
* Finding rows by using a simple equality(=)
* Finding rows that meet two conditions(AND)
* Finding rows by using a comparison operator(>,<,>=,<=)
* Finding rows that meet any of two conditions(OR)
* Finding rows with the value between two values(BETWEEN……AND…..)
  + Range will include both values.
* Finding rows that have a value in a list of values(IN)
* Finding rows whose values contain a string(LIKE….wild Character)
  + The percent wildcard (%): any string of zero or more characters.
  + The underscore (\_) wildcard: any single character.
  + The [list of characters] wildcard: any single character within the specified set.
    - The square brackets with a list of characters e.g., [ABC] represents a single character that must be one of the characters specified in the list.
  + The [character-character]: any single character within the specified range.
    - The square brackets with a character range e.g., [A-C] represent a single character that must be within a specified range.
  + The [^]: any single character not within a list or a range.

To negate the result of the LIKE operator, you use the NOT operator as follows:

### COUNT

Count (\*) – count null values also.

Count (1) – count null values also.

Count (col1) – does not count null values.

Count (distinct col1) - Remove duplicate entries also does not count null values.

Count(all) - count not null value only.

### NULL

In the database world, NULL is used to indicate the absence of any data value. For example, at the time of recording the customer information, the email may be unknown, so it is recorded as NULL in the database.

Normally, the result of a logical expression is TRUE or FALSE. However, when NULL is involved in the logical evaluation, the result is UNKNOWN . This is called a three-valued logic: TRUE, FALSE, and UNKNOWN.

*The results of the following comparisons are UNKNOWN:  
NULL is equal to nothing, even NULL is not equal to NULL because each NULL could be different*

NULL = 0

NULL <> 0

NULL > 0

NULL = NULL

Result set will be empty, as phone=Null will give unknown. Instead, we should use IS null or IS not NULL operator.

SELECT

customer\_id,

first\_name,

last\_name,

phone

FROM

sales.customers

WHERE

phone = NULL

ORDER BY

first\_name,

last\_name;

## Join

https://stackoverflow.com/questions/5706437/whats-the-difference-between-inner-join-left-join-right-join-and-full-join

<https://www.techmixing.com/2019/04/sql-joins-tricky-interview-questions.html>

### difference between Join and Union

Join merge column of table, while union merges rows of table.

### Self-join

A self-join allows you to join a table to itself. It is useful for querying hierarchical data or comparing rows within the same table.

The employee column does not have Fabiola Jackson because of the [INNER JOIN](https://www.sqlservertutorial.net/sql-server-basics/sql-server-inner-join/) effect. If you replace the [INNER JOIN](https://www.sqlservertutorial.net/sql-server-basics/sql-server-inner-join/) clause by the [LEFT JOIN](https://www.sqlservertutorial.net/sql-server-basics/sql-server-left-join/) clause as shown in the following query, you will get the result set that includes Fabiola Jackson in the employee column:

### Cross Join

Unlike the [INNER JOIN](https://www.sqlservertutorial.net/sql-server-basics/sql-server-inner-join/) or [LEFT JOIN](https://www.sqlservertutorial.net/sql-server-basics/sql-server-left-join/), the cross join does not establish a relationship between the joined tables. And return Cartesian product

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Natural Join/Inner join | Self Join | Equi Join | Left Outer Join | Right Outer Join | Full Join | Cross Join |
| On | Table with itself | Any attribute can be used in Join condition |  |  | Left  +common  +Right | Matrix multiplication |
| We do not check null |  |  | If null present in left table, as it has to come , so right column will be null |  |  |  |
| salary of emp  From each department | Select employee who are manager |  |  |  |  |  |

create table B(col1 nvarchar null)

create table C(col1 nvarchar null)

insert into B values(1)

insert into B values(1)

insert into B values(0)

insert into B values(Null)

Insert into C values(1)

Insert into C values(0)

Insert into C values(Null)

Insert into C values(Null)

select \* from [dbo].[B]

select \* from [dbo].[C]

----inner join

--select col1, col2

--from [dbo].[B] b join [dbo].[C] c

--on b.Col1 = c.Col2

--left join if right table dont have null what will be the output

--select col1, col2

--from [dbo].[B] b left join [dbo].[C] c

--on b.Col1 = c.Col2

select col1, col2

from [dbo].[B] b right join [dbo].[C] c

on b.Col1 = c.Col2

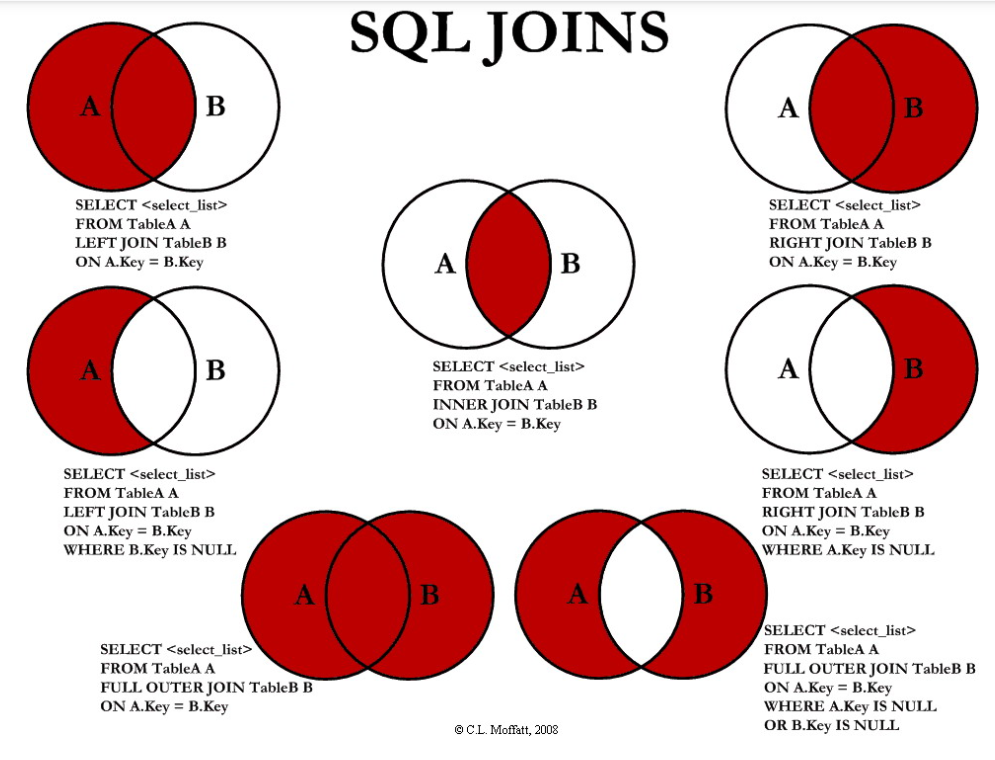
select col1, col2

from [dbo].[B] b full join [dbo].[C] c

on b.Col1 = c.Col2

select col1, col2

from [dbo].[B] b ,[dbo].[C] c



## Grouping data

### Group by

* In practice, the GROUP BY clause is often used with [aggregate functions](https://www.sqlservertutorial.net/sql-server-aggregate-functions/) for generating summary reports.
* The GROUP BY clause arranges rows into groups and an aggregate function returns the summary (count, min, max, average, sum, etc.,) for each group.

### Having

* The HAVING clause is often used with the [GROUP BY](https://www.sqlservertutorial.net/sql-server-basics/sql-server-group-by/) clause to filter groups based on a specified list of conditions.
* In this syntax, the [GROUP BY](https://www.sqlservertutorial.net/sql-server-basics/sql-server-group-by/) clause summarizes the rows into groups and the HAVING clause applies one or more conditions to these groups. Only groups that make the conditions evaluate to TRUE are included in the result. In other words, the groups for which the condition evaluates to  FALSE or UNKNOWN are filtered out.
* Because SQL Server processes the HAVING clause after the GROUP BY clause, you cannot refer to the [aggregate function](https://www.sqlservertutorial.net/sql-server-aggregate-functions/) specified in the select list by using the [column alias](https://www.sqlservertutorial.net/sql-server-basics/sql-server-alias/). The following query will fail:

SELECT

column\_name1,

column\_name2,

aggregate\_function (column\_name3) column\_alias

FROM

table\_name

GROUP BY

column\_name1,

column\_name2

HAVING

column\_alias > value;

SELECT

order\_id,

SUM (

quantity \* list\_price \* (1 - discount)

) net\_value

FROM

sales.order\_items

GROUP BY

order\_id

HAVING

SUM (

quantity \* list\_price \* (1 - discount)

) > 20000

ORDER BY

net\_value;

### Grouping set

This is used to analyse multiple groups.

SELECT

brand,

category,

SUM (sales) sales

FROM

sales.sales\_summary

GROUP BY

brand,

category

UNION ALL

SELECT

brand,

NULL,

SUM (sales) sales

FROM

sales.sales\_summary

GROUP BY

brand

UNION ALL

SELECT

NULL,

category,

SUM (sales) sales

FROM

sales.sales\_summary

GROUP BY

category

UNION ALL

SELECT

NULL,

NULL,

SUM (sales)

FROM

sales.sales\_summary

ORDER BY brand, category;

However, it has two major problems:

1. The query is quite lengthy.
2. The query is slow because SQL Server needs to execute four subqueries and combines the result sets into a single one.

To fix these problems, SQL Server provides a subclause of the [GROUP BY](https://www.sqlservertutorial.net/sql-server-basics/sql-server-group-by/) clause called GROUPING SETS.

The GROUPING SETS defines multiple grouping sets in the same query. The following shows the general syntax of the GROUPING SETS:

**SELECT**

**column1,**

**column2,**

**aggregate\_function (column3)**

**FROM**

**table\_name**

**GROUP BY**

**GROUPING SETS (**

**(column1, column2),**

**(column1),**

**(column2),**

**()**

**);**

#### Grouping Function

SELECT

GROUPING(brand) grouping\_brand,

GROUPING(category) grouping\_category,

brand,

category,

SUM (sales) sales

FROM

sales.sales\_summary

GROUP BY

GROUPING SETS (

(brand, category),

(brand),

(category),

()

)

ORDER BY

brand,

category;

 1 means that the sales amount is aggregated by brand, that means brand is aggregated to 1 row and category is used for creating groups.

0 means that the sales amount is not aggregated by brand. But grouped on brand

### Cube

The CUBE is a subclause of the [GROUP BY](https://www.sqlservertutorial.net/sql-server-basics/sql-server-group-by/) clause that allows you to generate multiple grouping sets. The following illustrates the general syntax of the CUBE:

SELECT

d1,

d2,

d3,

aggregate\_function (c4)

FROM

table\_name

GROUP BY

CUBE (d1, d2, d3);

### Roll Up

(d1, d2, d3)

(d1, d2)

(d1)

()

And the ROLLUP(d1,d2,d3) creates only four grouping sets, assuming the hierarchy d1 > d2 > d3, as follows:

The ROLLUP is commonly used to calculate the aggregates of hierarchical data such as sales by year > quarter > month.

# Query optimization

Optimization plan

Where is the error

How to fix it

From 🡪join🡪 where🡪 group by 🡪 having🡪 select🡪 order by

Set statistics io,time on

1. Using execution time check the logical read and physical read
   1. Logical read- virtual input output , like after the data is loaded in the system
   2. Physical read is the one time read . this occur one time only when we are doing initial load.
2. What we will look into an execution plan.
   1. Spooling –(temporary save of data to make compute easy)
      1. Lazy spool
         1. It mean solve a problem , now if there is no problem and we are telling compiler to solve the problem then Its bad.
         2. It happens when there is a duplicate aggregation.(group by , sum etc)
         3. It happens when recursion happen like recursive CTE . you can not avoid it(so leave it)
         4. Check why query is doing duplicate aggregation and try to fix it.
            1. Like corelated query ,  
               select \*  
               from table  
               where col in ( select max(age) from table group by col2)

Now for each run of outer query its doing aggreagation again and again . so this will cause an lazy spool. To fix this you can use temp table or cte and then use join to get the data.

* 1. Hash match
     1. Its mean unsorted data.
     2. Order by clause is expensive , but when we do group by then it first make it order by and then do grouping.
     3. Create index to fix this hash match will change to join
     4. Some time index is already exist ,but query wont utilize it . it happens because we are using expression to calculate, like year(2019)
  2. Key lookup
     1. If index does not include some columns, then it will again go to table and

Fetch data from table

* + 1. If you have only one column missing from index then you can take a chance .or else don’t take it , it will make index miserable.
  1. Bad views
     1. It a stored query.
     2. We are writing a procedure , and we are using a view to join. Now view is missing one column, then we are joiinng query with that table to fetch the column .
  2. Un necessary subqueries
  3. Comparing datetime with date
  4. Index scan
  5. Index seek
  6. Sargebility

# INDEX

* 1. Scan
  2. Seek

1. Indexes works on B tree.
2. Scan mean traverse through all the leaf node, and search for the data
3. Seek mean look for the path using the parant child logics
4. When we use expression in where clause like where month(invoice date) = may . then this expression will remove the benefit of index. This problem called searchabilty . search ability.

Index | row address

1001 | row 5

1002 | row 6

## Commands

1. Create index index\_name on table(col asc|desc)
2. Sp\_helpindex tablename------------This will return all the indexes on table
3. Drop index tablename indexname

## Type of indexes

1. Clustered index
2. Non-clustered index
3. Unique
4. Filtered
5. XM\L
6. Full text
7. Spatial
8. Index with included column
9. Index with calculated column

### Clustered index

1. It determines the physical order of data in the table
2. When we create PK in table , clustered index automatically gets created
3. It’s a clustered unique index
4. Determining physical order mean , if I insert the row in jumbled order than because of clustered index , row will itself change there place while inserting

2

1

5

3

After clustred index

2

Insert 1

1

2

Insert 5

1

2

3

Insert 3

1

2

3

5

A table can have only one clustered index. Because if we create one more clustered index , the row order of table will get messed up.

A clustered index can contain multiple column in it . this called composite index.

### Non- Clustered index-

* + - 1. Index stored saperately
      2. Data stored saperately
      3. Like book index, clustered index are like dictionary or telephone directory
      4. It require additional space , but clustered does not need
      5. Clustred determine table row insertion order , but no n clustered does not
      6. Clustered are faster than non clustered, as non clusted first look into index table than find that row in table
      7. Only 1 clustered index in table , but more than one clustered index

### Unique index

Enforece uniqueness in key

Unique index gets created when we create a unique key constraint