**Relational Database Management System(RDMS)**

**Normalization:** Normalization is the process of organizing the data to avoid duplicates and redundancy.

**Data Types:** Character, Numeric, Date and Time, and Binary(varbinary(n)).

**SQL Statements**

**Data Definition Language:** Create, Alter, Drop.

**Data Manipulation Language**: Select, Insert, Update, Delete.

**Data Control Language:** Grant, Revoke.

**Transaction Control Language:** Commit, Rollback, Save point.

**Create a Database**

USE master;

GO

CREATE DATABASE Sales

ON

( NAME = Sales\_dat,

FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\MSSQL\DATA\saledat.mdf',

SIZE = 10,

MAXSIZE = 50,

FILEGROWTH = 5 )

LOG ON

( NAME = Sales\_log,

FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\MSSQL\DATA\salelog.ldf',

SIZE = 5MB,

MAXSIZE = 25MB,

FILEGROWTH = 5MB );

GO

**Create a Table**

Use Database name

CREATE TABLE Test1

( empid int primary key,

L\_name varchar(50) NOT NULL,

F\_name varchar(50) NOT NULL

)

**Alter Table**

ALTER TABLE (Name) ADD (Column Name) dtype

ALTER TABLE (Name) DROP Column (Column Name)

Also you alter dtype of the columns.

**Drop Table**

DROP TABLE <Table Name>

**Various Constraints**

**NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK.**

CREATE TABLE emp

(

emp\_id INT PRIMARY KEY,

L\_NAME VARCHAR(50), NOT NULL OR

CONSTRAINT emp\_pk PRIMARY KEY (emp\_id) (to create multiple primary key use this)

);

emp-pk is an constant and it should be unique in the database.

**Unique column can have NULL values, but primary key can’t.**

**How to create two related table like parent and child table.**

CREATE TABLE product

( prod\_id int primary key,

Prod\_name varchar(50) not null,

Category varchar(50)

);

CREATE TABLE orders

( order\_id int primary key,

Prod\_id int not null,

Quantity int,

CONSTRAINT fk\_product\_id

FOREIGN KEY (prod\_id)

REFERENCES product (prod\_id)

ON DELETE CASCADE(when you delete column in parent… child also deletes use this key word)

);

**Creating a VIEW**

A view is named, derived, virtual table. A view takes the output of a query and treats it as a table.

CREATE VIEW <NAME> AS SELECT \* from <table name>.

**TABLE COMMANDS**

Insert statements :

USE <db\_name)

Insert into <table\_name) (column1, column2..) values (val1, val2…);

**You can also insert data from another table into another table.**

Update Statements

UPDATE <table\_name> SET <field name> = Value where <Condition>;

Delete Statements

DELETE from <table name> where <condition>

Select Statements

SELECT <fields…> from <table name> where <conditions>;

Order By : Used along with where clause to display the specified column in ascending order or descending order.

**Filtering: Logic operators[AND, OR, NOT]**

**Filtering: Comparison Operators[>, <, !=, <>, >=, Between, LIKE, IN, NOT IN]**

**$, ^ for string comparison.**

**[= operator don’t work with NULL, You have to use IS NULL to drifter column have NULL values]**

**CASE Expression**

SELECT (CASE When column\_name = “condition,abc” then result1 else 0 end )as abc ,

(Case When column name =” condition 2” then result2 else 0 end )as abc

From table name ;

SELECT N, CASE WHEN P IS NULL THEN 'Root'

WHEN N IN (SELECT P FROM BST) THEN 'Inner'

ELSE 'Leaf' **END AS Nodes**

FROM BST

ORDER BY N

**SQL JOINS**

SELF JOIN : A table can be joined to itself in a self join.

INNER JOIN : Inner join fetches records that have mating values in both the tables.

SELECT < column\_name>

From <table name> as t1

Inner join <table name 2> as t2

ON t1.column name = t2.column name;

LEFT OUTER JOIN : Return the rows to the left, even if there are now rows on the right of joins. If there are no matching rows on right table, it returns NULL values for that particular row.

SELECT < column\_name>

From <table name> as t1

Left outer join <table name 2> as t2

ON t1.column name = t2.column name;

RIGHT OUTER JOIN :Opposite to above.

FULL OUTER JOIN: Return all the rows, If no matching values it returns NULL values.

CROSS JOIN /CARTESIAN PRODUCT : Displays all the rows and all the columns of the tables. This is like multiplying all the rows of table with all the rows of table b.

**GROUP BY :** groups the rows that have same values into a summary rows. Group by statement is used when we make use of aggregate functions. It is like if you are using group by it's mandatory to aggregate functions like AVG(), SUM(), MIN(). Also should include all the columns used in select statement in group by clause.

**HAVING** : Having clause was added to SQL because the where keyword can’t be used with aggregate functions. It is used along with group by clause with aggregate function as filter.

SELECT col\_1, function\_name(col\_2)

FROM tablename

WHERE condition

GROUP BY column1, column2

HAVING Condition

ORDER BY column1, column2;

Now if we need to display the departments where the sum of salaries is 50,000 or more. In this condition, we will use HAVING Clause.

SELECT Department, sum(Salary) as Salary

FROM employee

GROUP BY department

HAVING SUM(Salary) >= 50000;

**SQL BUILT-in FUNCTIONS :** To calculate and manipulate data.

String Functions, Conversion Functions, Logical Functions, Math Functions, Aggregate Functions, Date Functions

**Conversion Function:** Conversion function covert expression from one data type to another.

[CAST, CONVERT, PARSE, TRY\_CAST, TRY\_CONVERT, TRY\_PARSE]

Logical Function : Used to display one of the several values based on condition.

[CHOOSE, IFF]

CHOOSE (index, val 1, val 2, val 3)include

IFF(Boolean\_exp, ‘TRUE’, ‘FALSE’)

**MATH Functions :**

[ABS, RAND, EXP, ROUND, FLOOR, SQRT, CEILING, SQUARE, POWER]

**STRING Function**: Operation on string input value and return string or numeric value.

[ LTRIM, RTRIM, CHAR, STR, CONCAT, CHARINDEX,LEFT, RIGHT, UPPER, LEN, SUBSTRING, REPLACE]

**Aggregate Function:**

[MIN, MAX, AVG]

**DATE TIME FUNCTION:**

[SYSDATETIME, CURRENT\_TIMTSTAMP, DATETIME, DATEPART, DATEDIFF, DATEADD]

**STORED PROCEDURES**

A stored procedure is a set of SQL statements with a name, that has been created and stored in the database.

Store procedures can be defined as the set of logical group of sql statements which are grouped to perform a specific task. Like a function in other programming language.

CREATE PROCEDURE

Schema\_name.procedure\_name (

@parameter\_name parameter\_data\_type[OUT])

AS

BEGIN

<SQL statements >

END;

**To execute the stored procedure**

Exec stored \_rocedure\_name @parameter\_name=’Value’ → this will give result.

**User Defined Function**

1. Scalar Function - return only single value.

CREATE FUNCTION

Shema\_name.function\_name (@parameter\_name parameter\_data\_type)

RETURNS return\_data\_type

AS

BEGIN

RETURN <SQL statement>;

END

1. Inline Table valued function -Return a whole table.

CREATE FUNCTION

Shema\_name.function\_name (@parameter\_name parameter\_data\_type)

RETURNS TABLE

AS

RETURN

(

<SQL statement>

);

GO

**TRIGGERS**

A trigger is a kind of stored procedure that automatically executes when an event occurs in the database.

CRETE TRIGGER

Shema\_name.trigger\_name

ON TABLE | database

FOR | AFTER | INSTEAD OF

[INSET],[UPDATE],[DELETE]

AS

SQl\_statement.

There are DDL and DML triggers.

**SQL INDEX**

Indexes are used to retrieve data from the database more quickly than otherwise.

CREATE INDEX index\_name

On TABLE(column1, column2);

**WINDOW FUNCTIONS[RANK(), DenseRank(), Rownumber()]**

Window function perform a calculation across a set of table rows that are somehow related to current row.

Window function apply aggregate and ranking function over a particular window (set of rows)

SELECT column\_name,

window\_function(column name) OVER (partition by column\_name ORDER\_BY column\_name)

AS column\_new\_name

From table.

window\_function(Avgerage,

Window functions applies aggregate and ranking functions over a particular window (set of rows). OVER clause is used with window functions to define that window. OVER clause does two things :

* Partitions rows into form set of rows. (PARTITION BY clause is used)
* Orders rows within those partitions into a particular order. (ORDER BY clause is used)

SELECT coulmn\_name1,

window\_function(cloumn\_name2)

OVER([PARTITION BY column\_name1] [ORDER BY column\_name3]) AS new\_column

FROM table\_name;

There is no mandate that you need to use both the partition and order inside over clause.

* **RANK() –**   
  As the name suggests, the rank function assigns rank to all the rows within every partition. Rank is assigned such that rank 1 given to the first row and rows having same value are assigned same rank. For the next rank after two same rank values, one rank value will be skipped.
* **DENSE\_RANK() –**   
  It assigns rank to each row within partition. Just like rank function first row is assigned rank 1 and rows having same value have same rank. The difference between RANK() and DENSE\_RANK() is that in DENSE\_RANK(), for the next rank after two same rank, consecutive integer is used, no rank is skipped.
* **ROW\_NUMBER() –**   
  It assigns consecutive integers to all the rows within partition. Within a partition, no two rows can have same row number

RANK- 1,1,3

Dense Rank- 1,1,2

Row Number - 1,2,3,4

**LEAD (next row) and LAG(previous row) Window function**

Lead and Lag function are useful when you want to compare current row data with previous row or next row. [Comparing data of different rows]

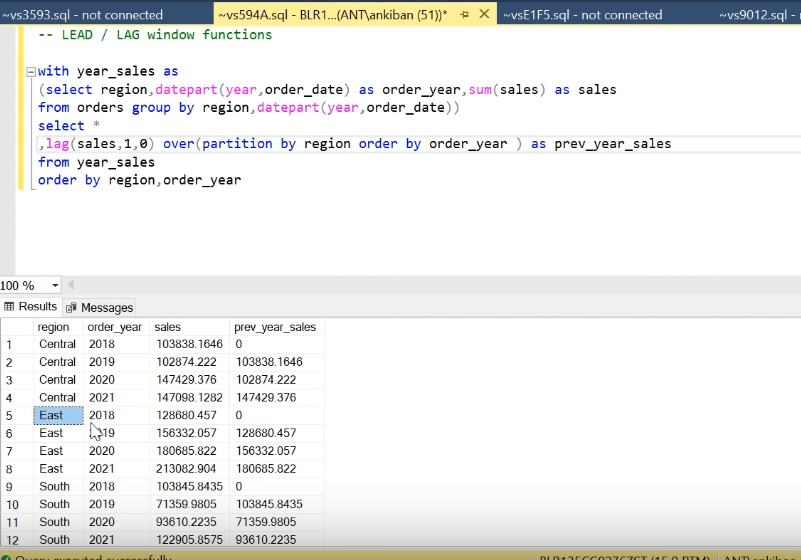
Select \*, lag (sales, 1) over (order by order\_year) as prev\_year\_sales

From yearly\_sales

Order by order\_year.

**In lag the 2nd argument may be 1|2|3 and 3rd argument is optional. If you give nothing it take null. Else you can give lag(sales, 1, 0) for make first row as 0 instead of null.**

**Here we did with year, What if there is region/country like that. Then you can make use of partition key to separate the regions.**

****

----------------------------------------------------------------------------------------------------------------------------

**SQL LIMIT :** Limit the record to display.

SELECT

select\_list

FROM

table\_name

LIMIT [offset,] row\_count; (for 2nd rank helpful)

[Limit command](https://www.plus2net.com/sql_tutorial/sql_limit.php) will pick up one record staring after the first record so we will get the second highest. If we want to get the **first record** or the highest mark student then we have to start from 0 record and display one. Like this

SELECT \* FROM `student` where class='Six' ORDER BY mark desc LIMIT 0,1

----------------------------another solution-------------------------------

**SELECT** max(marks)**FROM** Student **GROUP** **BY** sub\_name **WHERE**

marks **NOT** **IN** (**SELECT** max(marks) **FROM** Student **GROUP** **BY** sub\_name);

====================================================================

Count (\*), count(1), count(-1) all are same. They all count, even NULL values.

Count(‘Column Name’) → Only count, not null rows/values.

=================================================================

Examples

select case

when (A+B <= C) or (A+C <= B) or (B+C <= A) then 'Not A Triangle'

when (A=B and B=C) then 'Equilateral'

when (A = B) or (B=C) or (A=C) then 'Isosceles'

else 'Scalene'

end as triangles\_types

from TRIANGLES;

================================================================

Founder

|

Leader Manager

|

Senior Manager

|

Manager

|

Employee

SELECT c.company\_code,c.founder,

count(distinct lm.lead\_manager\_code),

count(distinct sm.senior\_manager\_code),

count(distinct m.manager\_code),

count(distinct e.employee\_code)

FROM Company c, Lead\_Manager lm, Senior\_Manager sm, Manager m, Employee e

WHERE

c.company\_code=lm.company\_code AND

lm.lead\_manager\_code=sm.lead\_manager\_code AND

sm.senior\_manager\_code=m.senior\_manager\_code AND

m.manager\_code=e.manager\_code

GROUP BY c.company\_code,c.founder

ORDER BY c.company\_code ASC

=================================

select merchant, sum(case when payment\_node = 'Cash' then amount else 0 end) as cash\_payment,

sum(case when payment\_mode ='Online' then amount else 0 end) as online\_amount

from payment\_data

group by merchant

============================

BST -

SELECT N,

CASE

WHEN P IS NULL THEN 'Root'

WHEN N IN (SELECT P FROM BST) THEN 'Inner'

ELSE 'Leaf'

END

AS Nodes

FROM BST

ORDER BY N

======================================================

Query the two cities in **STATION** with the shortest and longest *CITY* names, as well as their respective lengths (i.e.: number of characters in the name). If there is more than one smallest or largest city, choose the one that comes first when ordered alphabetically.

(

SELECT CITY, LENGTH(CITY)

FROM STATION

ORDER BY LENGTH(CITY), CITY

LIMIT 1

)

UNION

(

SELECT CITY, LENGTH(CITY)

FROM STATION

ORDER BY LENGTH(CITY) DESC, CITY

LIMIT 1

)

==============================================================

Query the list of *CITY* names from **STATION** that either do not start with vowels or do not end with vowels. Your result cannot contain duplicates.

.\* - all columns from the table/alias named”(dot start means)

^ - symbol used for pattern. In [%MATCHES](https://docs.intersystems.com/irislatest/csp/docbook/DocBook.UI.Page.cls?KEY=RSQL_matches) pattern string a NOT character. For example, [^abc].

SELECT DISTINCT city FROM station WHERE city RLIKE '^[^aeiouAEIOU].\*|.\*[^AEIOUaeiou]$';

=====================================================================

Concatenation of 2 columns

SELECT

CONCAT(first\_name, ' ', last\_name) AS full\_name

FROM patients;

==============================================

SELECT COUNT(\*) AS total\_patients

FROM patients

WHERE YEAR(birth\_date) = 2010;

=================================================

Count male and femain in table ..Nested

SELECT

(SELECT count(\*) FROM patients WHERE gender='M') AS male\_count,

(SELECT count(\*) FROM patients WHERE gender='F') AS female\_count;

======================================================

Show the city and the total number of patients in the city in the order from most to least patients.

SELECT

city,

COUNT(\*) AS num\_patients

FROM patients

GROUP BY city

ORDER BY num\_patients DESC;

===============================================================

Find the records who (caller id) called other person 1 st and last on that particular day.

**-- create a table**

**CREATE TABLE phonelog (**

**Callerid INTEGER not NULL,**

**Recipientid INTEGER NOT NULL,**

**Datecalled datetime NOT NULL**

**);**

**-- insert some values**

**insert into phonelog(Callerid, Recipientid, Datecalled)**

**values(1, 2, '2019-01-01 09:00:00.000'),**

**(1, 3, '2019-01-01 17:00:00.000'),**

**(1, 4, '2019-01-01 23:00:00.000'),**

**(2, 5, '2019-07-05 09:00:00.000'),**

**(2, 3, '2019-07-05 17:00:00.000'),**

**(2, 3, '2019-07-05 17:20:00.000'),**

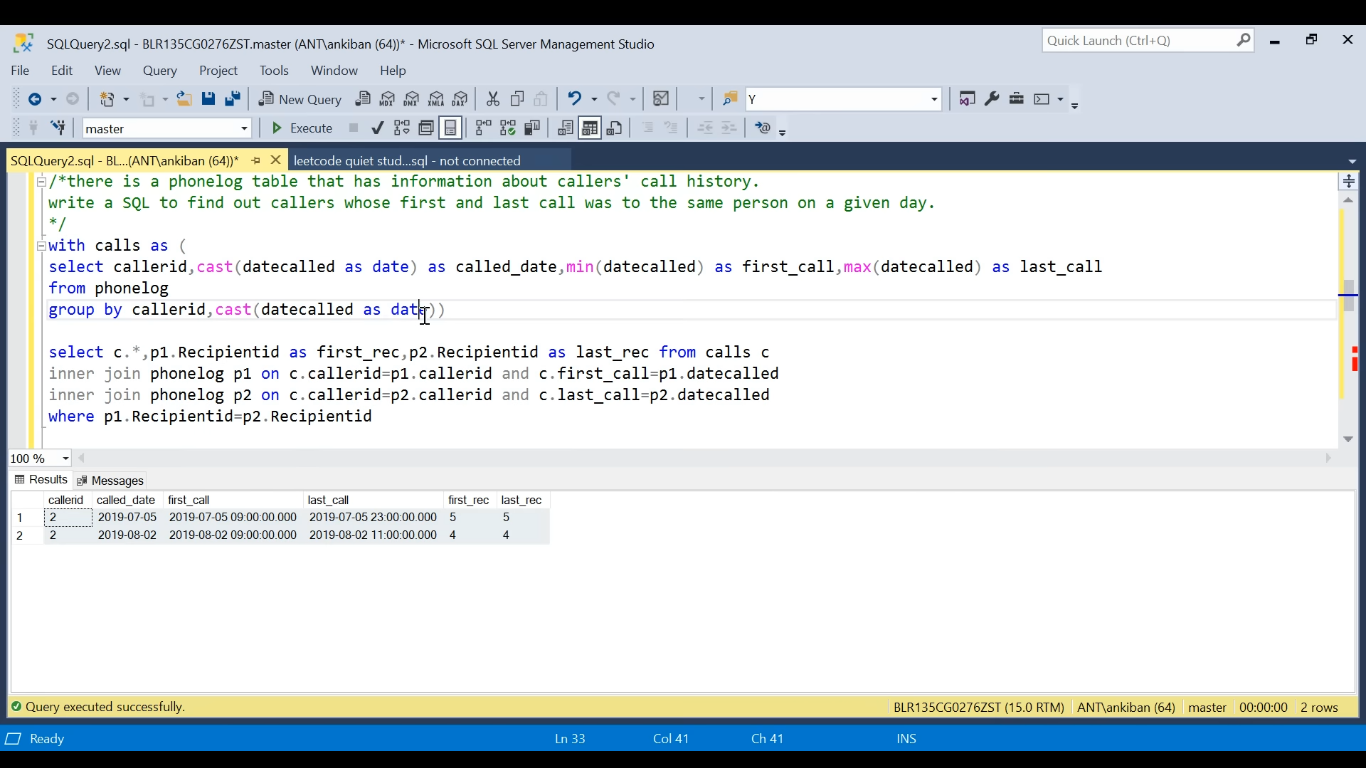
**(2, 5, '2019-07-05 23:00:00.000'),**

**(2, 3, '2019-08-01 09:00:00.000'),**

**(2, 3, '2019-08-01 17:00:00.000'),**

**(2, 5, '2019-08-01 19:30:00.000'),**

**(2, 4, '2019-08-02 09:00:00.000');**



First find min, max date called and group by callerid and datecalled.

use phone(db)

with calls as (select callerid, cast(datecalled as date), min(datecalled) as f\_call, max(datecalled) as l\_call from phonelog

group by callerid, cast(datecalled as date))

Then You have new calls table(caller id, first\_call, last\_call) and old phonelog table(3 column callerid, recipient id, datecalled)

Then join caller if of c with p1(table 1) callerid and first\_call with p1. Datecalled. Similarly with other join.

Then select p1.recipient fro phone log, and p2. You get first anf last recipients for the caller ids. Then in where condition recipient 1st equals last.

select c.\*,p1.recipientid as first\_r, p2.recipientid as last\_r from calls c

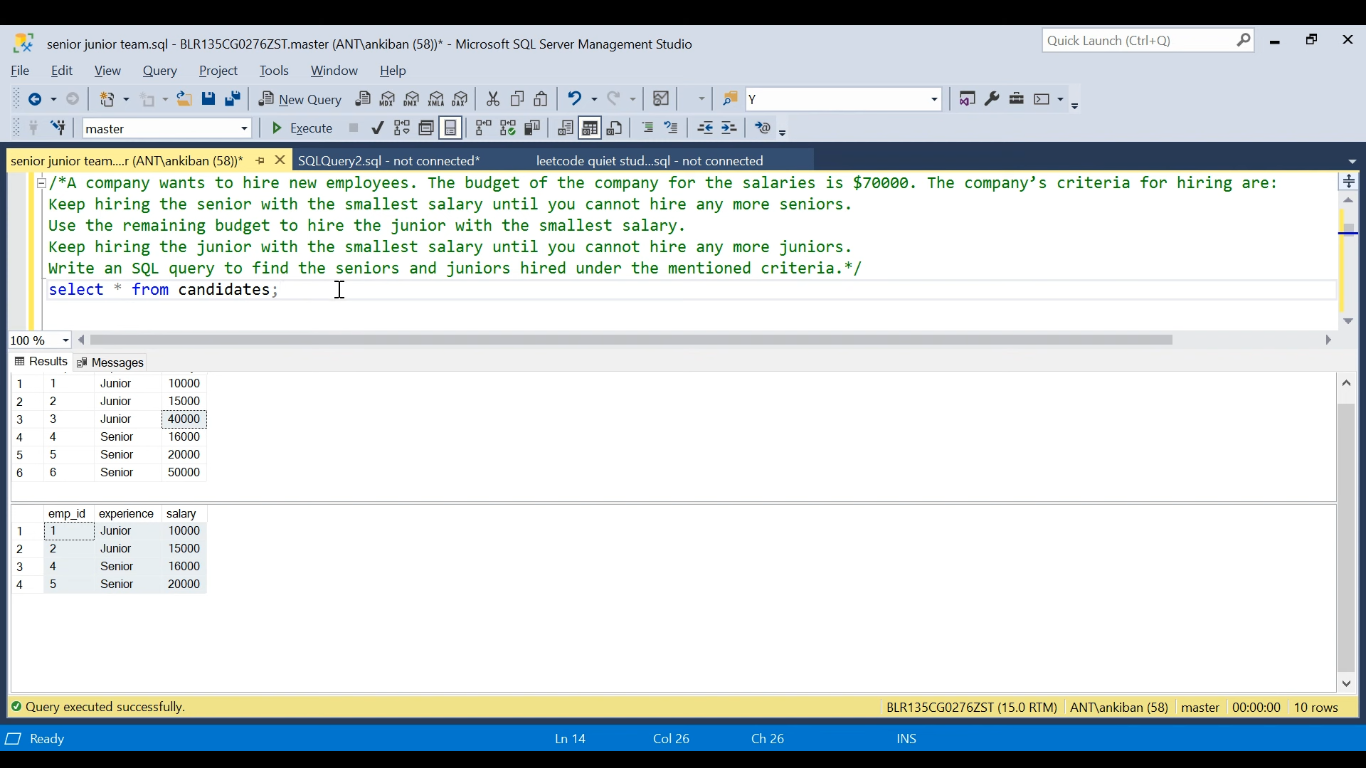
inner join phonelog p1 on p1.callerid=c.callerid and p1.datecalled=c.f\_call

inner join phonelog p2 on p2.callerid=c.callerid and p2.datecalled=c.l\_call

where p1.recipientid=p2.recipientid

==================================================================

CTE stands for **common table expression**. A CTE allows you to define a temporary named result set that available temporarily in the execution scope of a statement such as SELECT , INSERT , UPDATE , DELETE , or MERGE .



First using window function find the running sum.

use cand

with total\_sal as(select \*,sum(salary) over(partition by experience order by salary asc) as running\_sal

from candidates)

Then write condition for hiring senious without crossing 70k then store the table in senior(CTE)

,senior as(select \* from total\_sal where experience='senior' and running\_sal<=70000)

Then again hire junior with 70kminus sum of salary from above senior(CTE)

And union all.

select \* from total\_sal where experience='junior' and running\_sal<=70000-(select sum(salary) from senior)

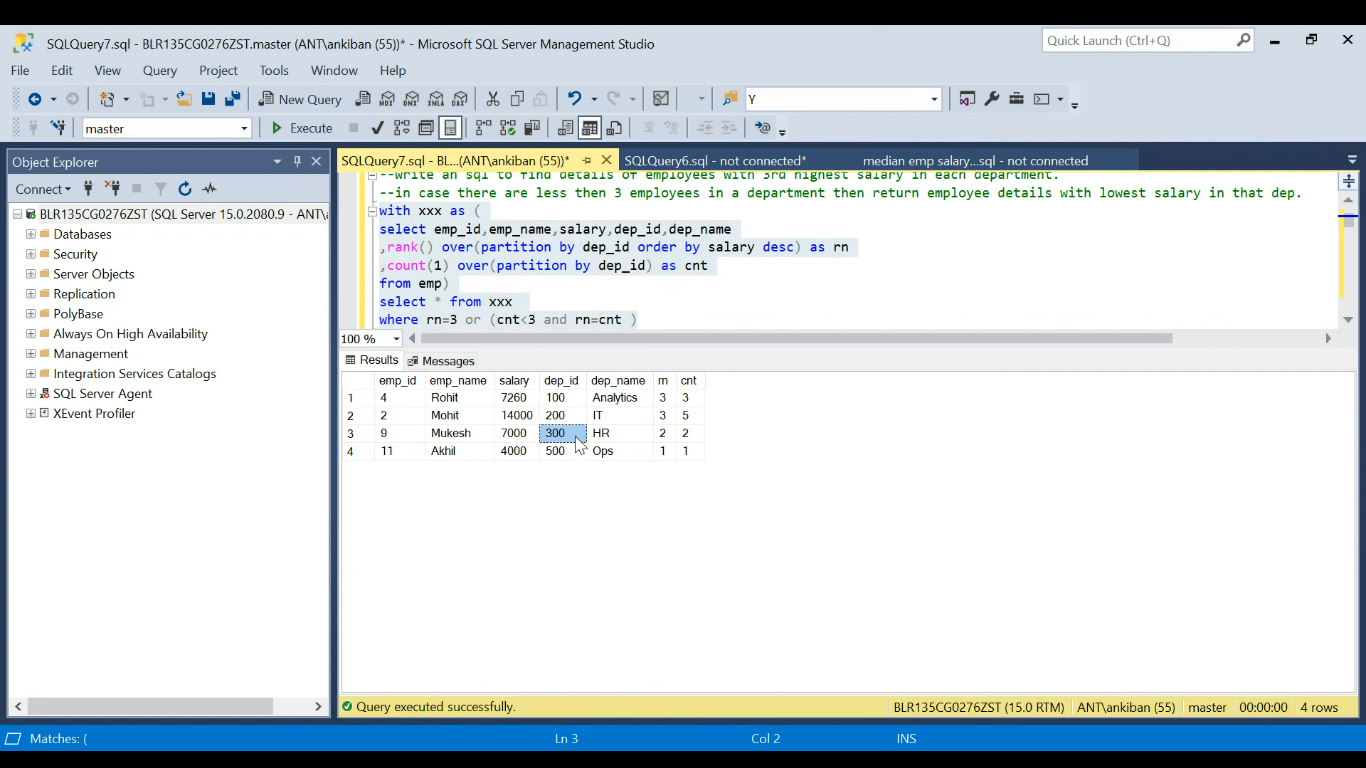
union all

select \* from senior

----------------------------------------------------------------

If duplicate values then use salary asc… rows between unbounded preceding and current row.

==========================================================================

****

**With abc as(Select emp\_id, emp\_name,salary, dep\_name,**

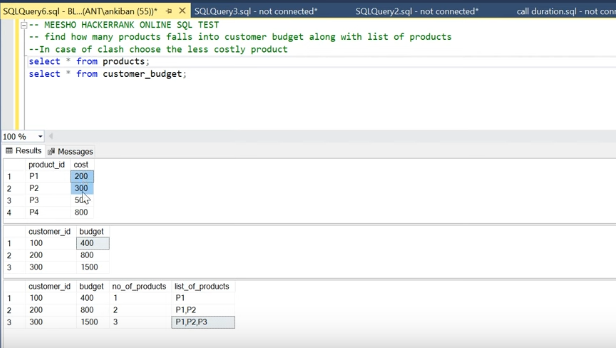
**rank() over(partition by dep\_id order by salary desc) as rank**

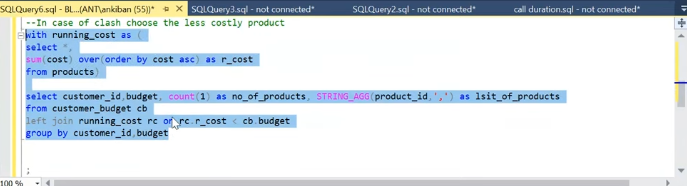
**,count(1) over(partition by dep\_id) as cnt**

**From emp)**

**Select \* from abc where rank=3 or(cnt<3 and rank = cnt)**

**=================================================================**

****

****

First sort the table based on cost and check for each customer id with budget, which all the products id comes in(p1,p2,p3,..)

First find running cost for product table.

With r\_cost as(Select \* from sum(cost) over (order by cost asc) as r\_cost

From products )

Then left join with customer table

Select customer\_id, budget, count(1) as no\_of\_products, STRING\_AGG(product\_id, ‘,’)as list\_of\_products from customer\_budget cb

Left join r\_cost rc **on rc.r\_cost < cb.customer\_budget(NO need not be always on equal sign)**

Group by customer\_id, budget;

====================================================================

SQL join when key are same/identical.(records)

Create table t1(id1 int);

Create table t2(id2 int);

Insert into t1(1);

Insert into t1(1);

Insert into t2(1);

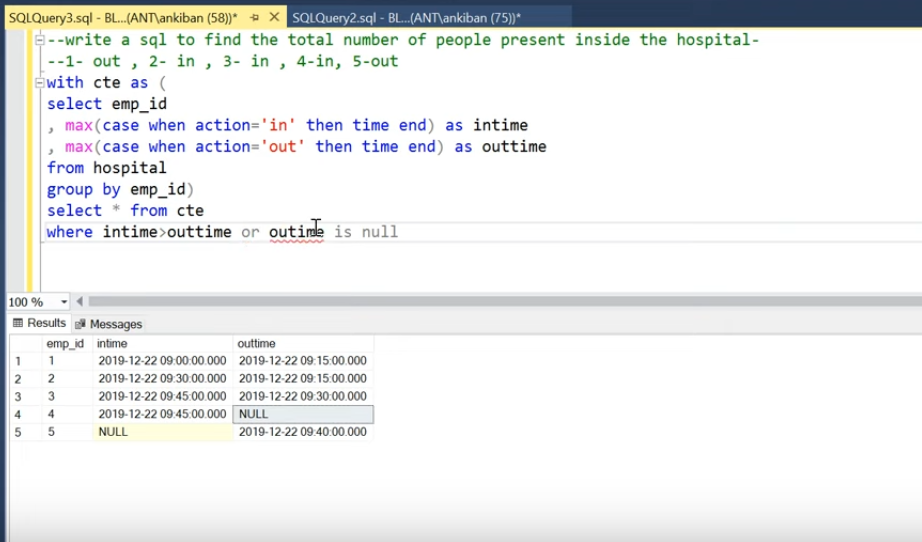
Insert into t2(1);

Insert into t(1);

Here i have crete a two tables with values 1, 1, in t2 an 1,1,1 in t2.

When you do inner, left, right, out join you get 6 records. As system cant differentiate, it will join each element of t1 with t2.

---------------------------------------------------------------------------------------------------------------------------



First using case statement separate the table column time into in time column and out time column. Then take max of in time and out time and group by employee id.

Then put to CTE and where intime>outtime ot out time is null.

===================================================================

<https://www.geeksforgeeks.org/difference-between-clustered-and-non-clustered-index/>

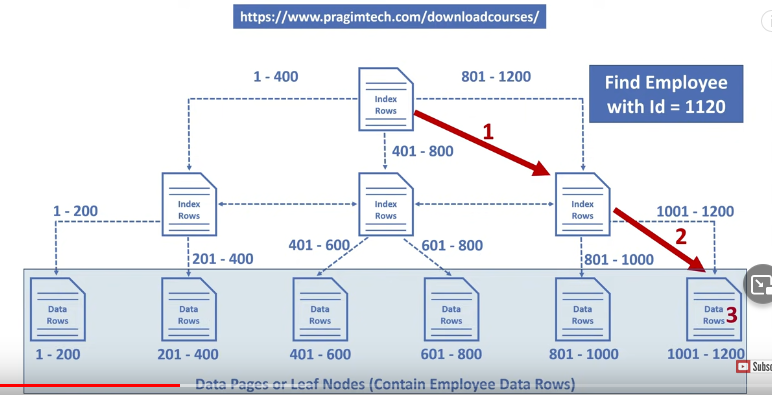
**1. Clustered Index :**

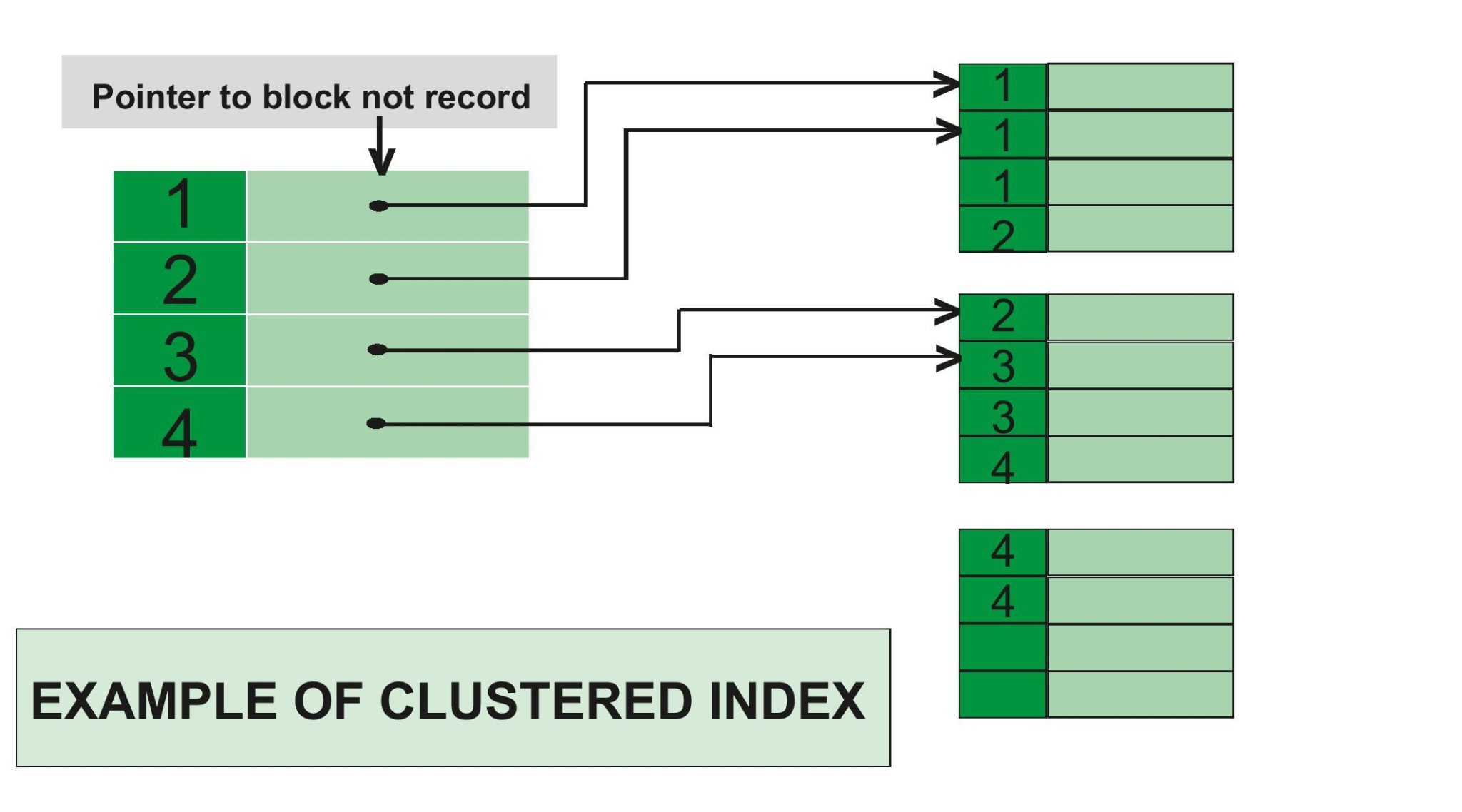
Clustered index is created only when both the following conditions satisfy –

1. The data or file, that you are moving into secondary memory should be in sequential or sorted order.
2. There should be a key value, meaning it can not have repeated values.

Whenever you apply clustered indexing in a table, it will perform sorting in that table only. You can create only one clustered index in a table like primary key.

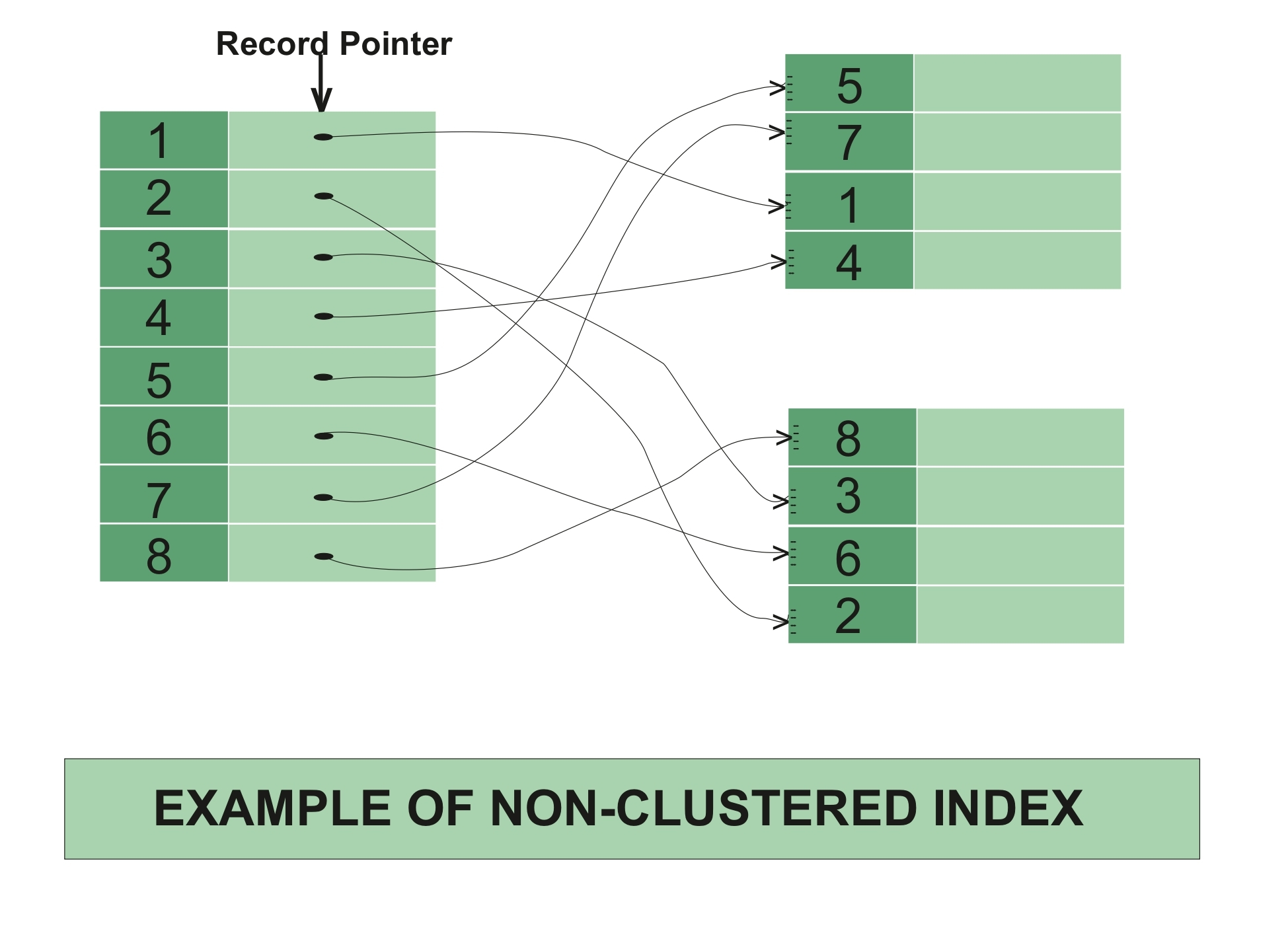
If you apply primary key to any column, then automatically it will become clustered index.

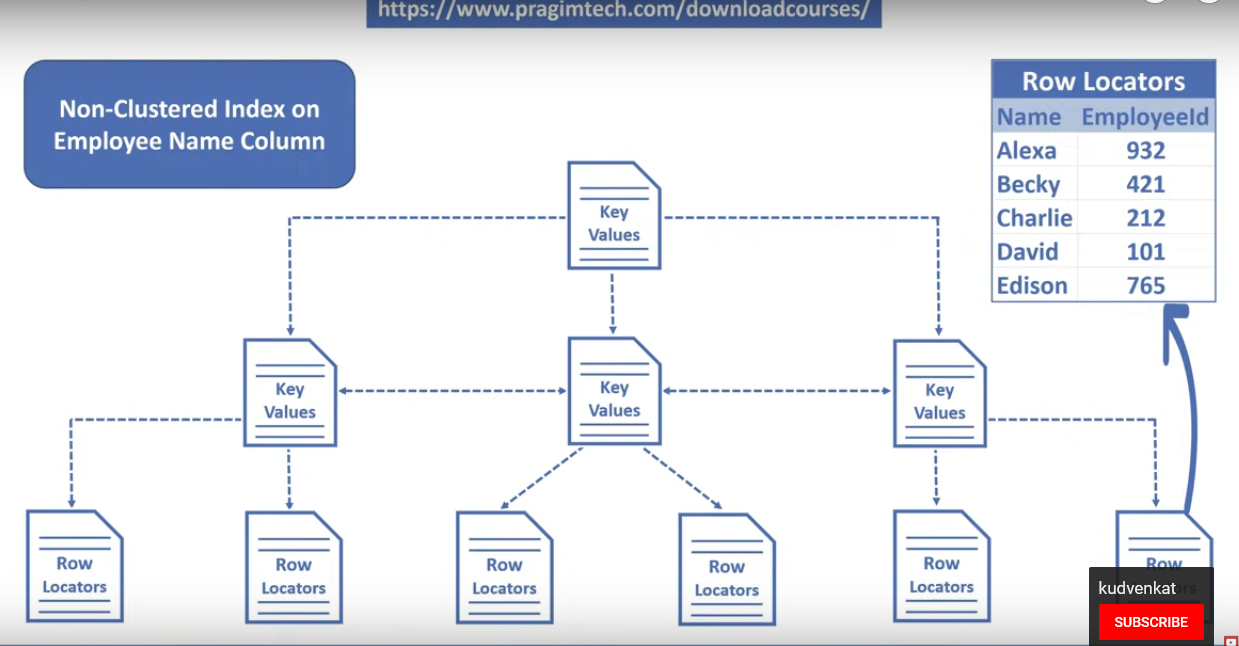


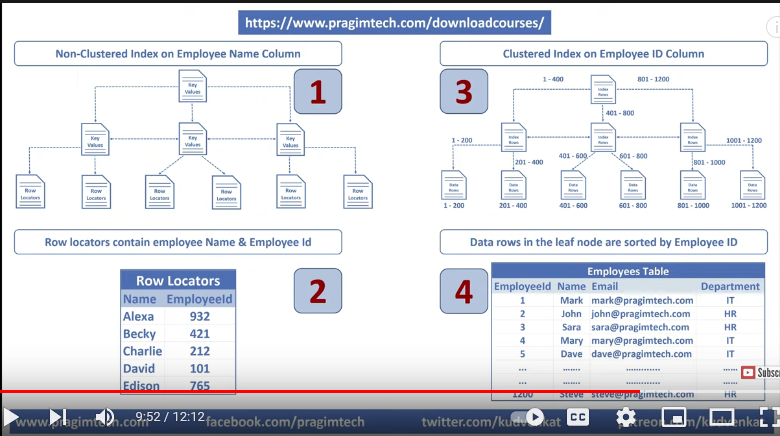


**2. Non-clustered Index :**

Non-Clustered Index is similar to the index of a book. The index of a book consists of a chapter name and page number, if you want to read any topic or chapter then you can directly go to that page by using index of that book.



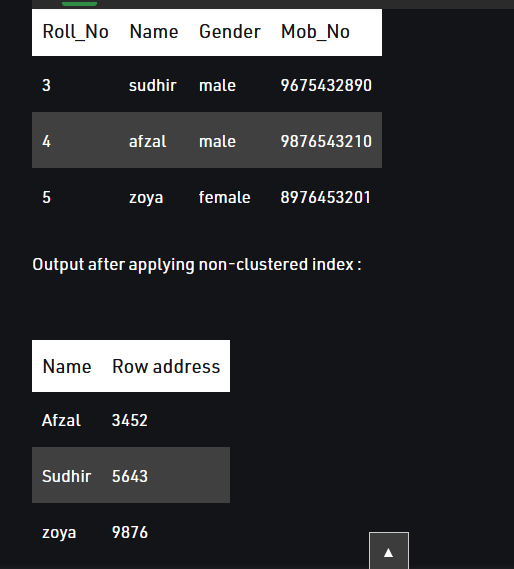




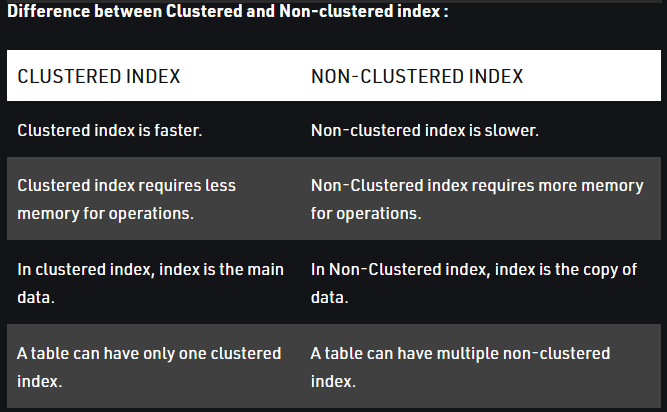
The data is stored in one place, and index is stored in another place. Since, the data and non-clustered index is stored separately, then you can have multiple non-clustered index in a table.

Here, roll no is a primary key, hence there is automatically a clustered index.

If we want to apply non-clustered index in NAME column (in ascending order), then the **new table** will be created for that column.



Row address is used because, if someone wants to search the data for sudhir, then by using the row address he/she will directly go to that row address and can fetch the data directly.

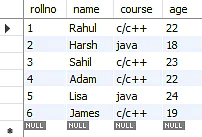


====================NOMALIZATION in SQL============================

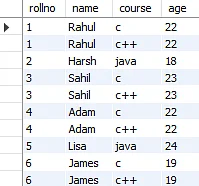
**Normalization:** Normalization is the process of organizing the data to avoid duplicates and redundancy.

## **1st Normal Form (1NF)**

Single cell cannot hold multiple values. It must hold only a single-valued attribute.



**INTO**

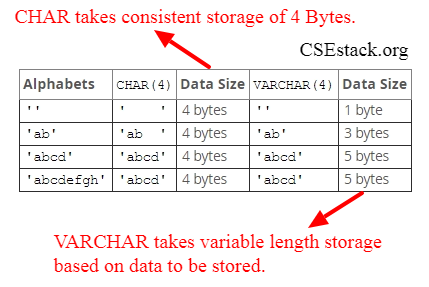


===========================================================

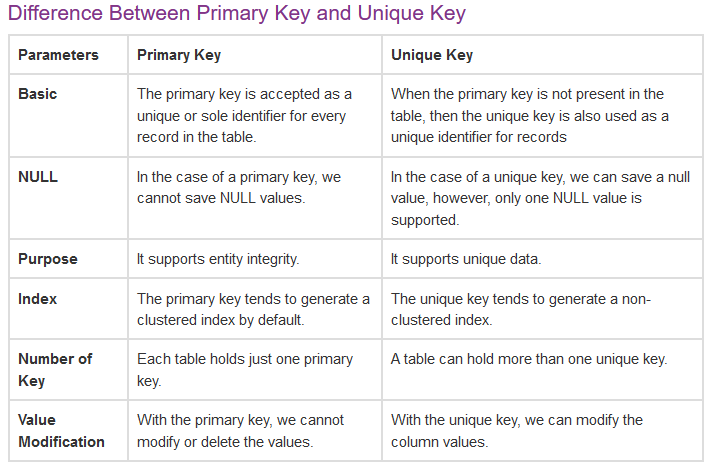
.CHAR vs VARCHAR →

CHAR - It is a datatype in SQL which is used to store character string of fixed length specified.

VARCHAR - It is a datatype in SQL which is used to store character string of variable length but a maximum of the set length specified.



==============PRIMARY KEY and UNIQUE KEY =================



============================================================

## **SQL Create Constraints**

Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

The following constraints are commonly used in SQL:

* [NOT NULL](https://www.w3schools.com/sql/sql_notnull.asp) - Ensures that a column cannot have a NULL value
* [UNIQUE](https://www.w3schools.com/sql/sql_unique.asp) - Ensures that all values in a column are different
* [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
* [FOREIGN KEY](https://www.w3schools.com/sql/sql_foreignkey.asp) - Prevents actions that would destroy links between tables
* [CHECK](https://www.w3schools.com/sql/sql_check.asp) - Ensures that the values in a column satisfies a specific condition
* [DEFAULT](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column if no value is specified
* [CREATE INDEX](https://www.w3schools.com/sql/sql_create_index.asp) - Used to create and retrieve data from the database very quickly

==================================================

**SQL script to remove duplicates from the table**

**with cte1 as ( select contact\_id, first\_name, last\_name, email, ROW\_NUMBER() over(partition by first\_name, last\_name, email order by first\_name, last\_name, email) as row\_num**

**from contact)**

**delete from cte1**

**where row\_num >1;**

**============================================================**

**What is difference between TRUNCATE and DELETE and DROP in SQL?**

#### **DELETE**

**The DELETE statement in SQL is a Data Manipulation Language(DML) Command. It is used to delete existing records from an existing table. We can delete a single record or multiple records depending on the condition specified in the query.**

**Note – Here we can use the “ROLLBACK” command to restore the tuple because it does not auto-commit.**

#### TRUNCATE

TRUNCATE Command is a Data Definition Language operation. It is used to remove all the records from a table. It deletes all the records from an existing table but not the table itself. The structure or schema of the table is preserved.

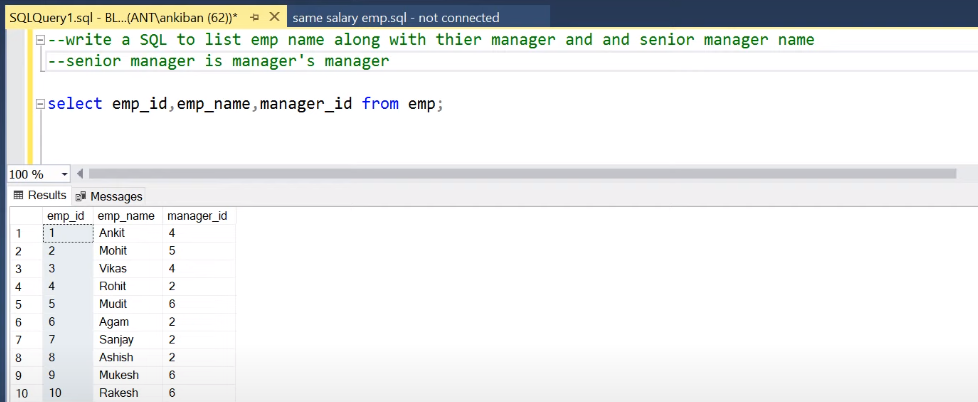
**Note – Here we can’t restore the tuples of the table by using the “ROLLBACK” command.**

**It is a Data Definition Language Command (DDL). It is used to drop the whole table. With the help of the “DROP” command we can drop (delete) the whole structure in one go i.e. it removes the named elements of the schema. By using this command the existence of the whole table is finished or say lost.**

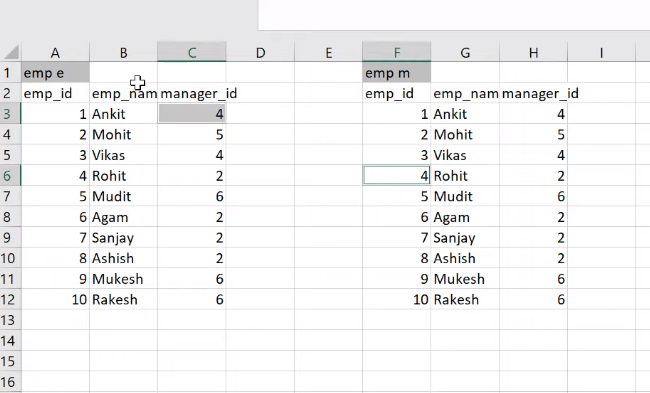
**Note –** Here we can’t restore the table by using the “ROLLBACK” command because it auto commits.

-------------------------------------------------------------------------------------------------------------------

Write sql to get manager and senior manger from emp table.



When same table columns are referencing to each other, you should always use self join.



Now you do **left join** (even if they don’t have manger, he should be in output). With table e and m.

