**SQL QUERIES (46pages are printed)**

**------------------------------------------------------------------------------------------------------------------**

**Use incedo**

**--------------------------------------------------------------------------------------------------------------------**

**select count (\*) from Cricketer\_ODI\_Statistics order by Half\_Century desc**

**---------------------------------------------------------------------------------------------------------------------**

**create table EmployeeDetails( empid varchar(50) not null primary key ,FullName varchar(60), ManagerId varchar(50),DateOfJoining date )**

**--------------------------------------------------------------------------------------------------------------------**

**insert into EmployeeDetails values ('121','John',321,'12-09-2019')**

**insert into EmployeeDetails values ('122','Keran',322,'12-09-2019')**

**insert into EmployeeDetails values ('123','Nik',323,'12-09-2019')**

**insert into EmployeeDetails values ('124','Olga',324,'10-04-2019')**

**insert into EmployeeDetails values ('125','Pamela',325,'01-09-2019')**

**insert into EmployeeDetails values ('126','Dimtry',326,'02-07-2019')**

**insert into EmployeeDetails values (127,'perry' ,327 , '2019-9-09')**

**insert into EmployeeDetails values (129,'perry' ,329 , '2019-9-09')**

**insert into EmployeeDetails values (128,'perry' ,328 , '2019-9-09')**

**---------------------------------------------------------------------------------------------------------------------**

**update EmployeeDetails set FullName= 'diyan' where empid=126**

**update EmployeeDetails set FullName= 'John Perry' where empid=121**

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

**select \* from EmployeeDetails**

empid FullName ManagerId DateOfJoining

121 John Perry 321 2019-12-09

122 keran kerry 322 2019-12-09

123 Nik 323 2019-12-09

124 Olga 324 2019-10-04

125 Pamela 325 2019-01-09

126 diyan 326 2019-02-07

127 herry 327 2019-09-09

128 perry 328 2019-09-09

129 perry 329 2019-09-09

**Note**

update command used for data manipulation from database and Alter command used for change the structure of table like alter table delete ,rename ,modify, drop table

**---------------------------------------------------------------------------------------------------------------------**

**create table EmployeSalary( empid varchar(50) foreign key (empid) references employeedetails(empid) ,project varchar(50),salary money )**

**--------------------------------------------------------------------------------------------------------------------**

insert into EmployeSalary values('121','p1',120000)

insert into EmployeSalary values('122','p1',320000)

insert into EmployeSalary values('123','p2',220000)

insert into EmployeSalary values('124','p2',420000)

insert into EmployeSalary values('125','p3',920000)

insert into EmployeSalary values('126','p3',190000)

insert into EmployeSalary values('127','p4',120000)

**---------------------------------------------------------------------------------------------------------------------**

**select \* from EmployeSalary**

**empid project salary**

121 p1 120000.00

122 p1 320000.00

123 p2 220000.00

124 p2 420000.00

125 p3 920000.00

126 p3 190000.00

127 p3 120000.00

128 p3 NULL

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

**insert into EmployeeDetails values (129,'perry' ,32 , '2019-9-09')**

**insert into EmployeSalary values (127,'p3', 120000)**

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

**Q1) Write a SQL query to fetch the count of employees working in project 'P1'.**

Ans select count( \*) from EmployeSalary where project ='P1'

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

Q2**) Write a SQL query to fetch employee names having salary greater than or equal to 10,000 and less than or equal 40,000**.

Ans :-select fullname from EmployeeDetails ed join EmployeSalary es on ed.empid= es.empid where salary between 100000 and 400000

**Ed,es are the alias**

**---------------------------------------------------------------------------------------------------------------------**

**Q3) Write a SQL query to fetch project-wise count of employees sorted by project's count in descending order**.

Ans

select count(\*), project from EmployeSalary group by project order by project desc

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

**Project wise means use group by and condition is that always need to use aggregate function when use group by clause**

**Note:** group by clause is executing by taking any coulmn name but condition is that whether in select statement must have any agggregate function

(**select count(managerid) from EmployeeDetails group by FullName)**or select statemnent must have column name which is grouped by that column

(**select ManagerId from EmployeeDetails group by ManagerId,FullName**).

**Note:**- whatever column u will add in select statement it must be either in aggregate function or group by clause other wise it will give error

Like:-

**select count(ManagerId) from EmployeeDetails group by ManagerId(give accurate output)**

**Error in this case :-**because fullname column not exist in group by means to say we have to take select column in group by clause otherwise throw error

**select count(ManagerId),fullname from EmployeeDetails group by ManagerId**

**Error Statement:-**

Msg 8120, Level 16, State 1, Line 38

Column 'EmployeeDetails.FullName' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.

**solution :**

**select count(ManagerId),fullname from EmployeeDetails group by ManagerId,FullName**

**O/P**

(No column name) fullname

1 Ander

1 diyan

1 herry

1 John Perry

1 keran kerry

1 Nik

1 olga

1 Olga

1 olga

1 Pamela

1 perry

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

**Ques.4**. **Write a query to fetch only the first name(string before space) from the FullName column of EmployeeDetails table.**

**Ans**. In this question, we are required to first fetch the location of the space character in the FullName field and then extract the first name out of the

FullName field. For finding the location we will use CHARINDEX in SQL SERVER and for fetching the string before space, we will use SUBSTRING method.

Charindex help to fetch the index

**Substring is a inbuilt function needs 3 arguments 1) expression ,2) starting Position is in int ,3)length is in int.**

**SELECT SUBSTRING(FullName, 0, 8) FROM EmployeeDetails;**

o/p

(No column name)

John Pe

keran k

Nik

Olga

Pamela

diyan

**select CHARINDEX('e',FullName) from EmployeeDetails**

o/p

(No column name)

7

2

0

0

4

0

Here it gives the index of e character wherever the e is present in the column of fullname

**Ans**

**SELECT SUBSTRING(FullName , 0, CHARINDEX(' ',FullName)) FROM EmployeeDetails;**

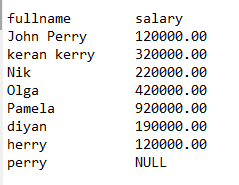
When we use it is with substring it means it will give u substring before this letter but in our question we need letter before space so provide space in

charindex query

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

**Q5) Write a query to fetch employee names and salary records. Return employee details even if the salary record is not present for the employee.**

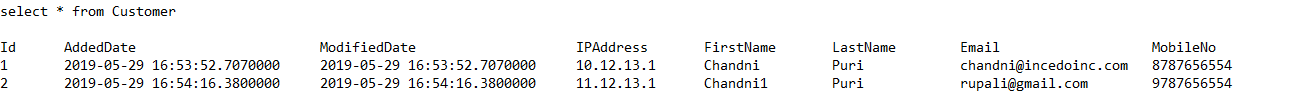
**select fullname, salary from EmployeeDetails join EmployeSalary on EmployeeDetails.empid=EmployeSalary.empid**



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

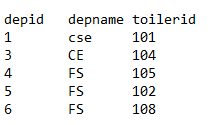
---------------------------------------------------------------------------------------------------------------------

**select \* from Customer**



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

1)**select \* from department**



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

2) **(join queries) on two tables which is customer and department table**

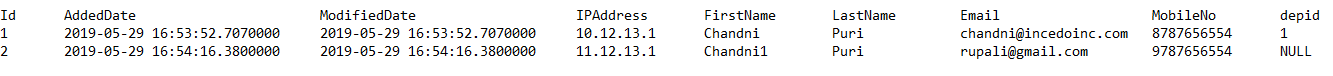
**select \* from Customer join department on Customer.Id=department.depid(give common entries from both table)**



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

3)

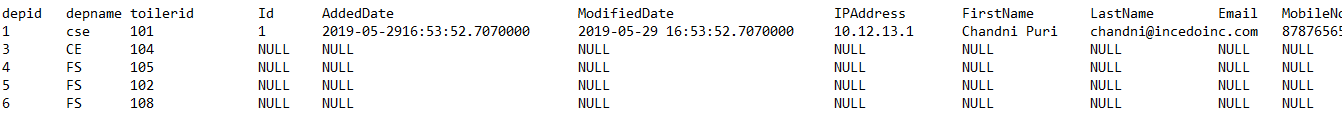
**select \* from Customer left join department on Customer.Id=department.depid**

****

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

4)

select \* from department **left join** Customer on Customer.Id=department.depid



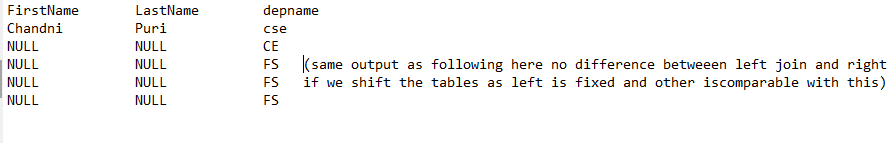
---------------------------------------------------------------------------------------------------------------------

5)

**select Customer.FirstName,Customer.LastName ,department.depname from department left join customer on Customer.Id=department.depid**

**or**

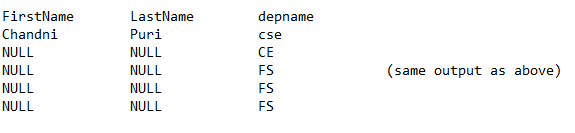
**select FirstName,LastName ,depname from department left join customer on Customer.Id=department.depid**

****

(same output as following here )no difference between left join and right if we shift the tables as left is fixed and other is comparable with this)

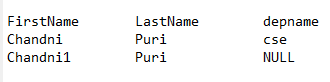
**6)**

**select FirstName,LastName ,depname from customer right join department on Customer.Id=department.depid**

****---------------------------------------------------------------------------------------------------------------------

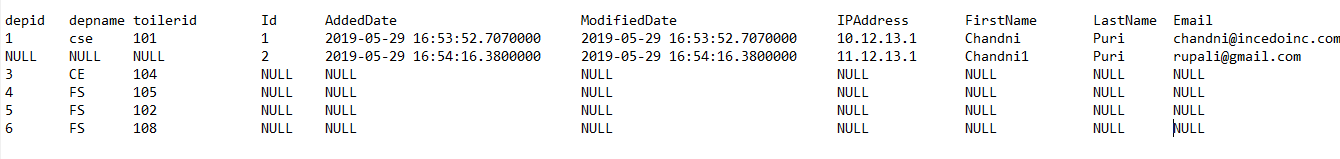
**7)**

select FirstName,LastName ,depname from department **right join** customer on Customer.Id=department.depid



**8)**

select \* from department **full outer join** Customer on Customer.Id=department.depid



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

**create table employeeTransportation( empid varchar(50) foreign key references employeedetails(empid) , allowance varchar(50),empaddress varchar(50))**

**---------------------------------------------------------------------------------------------------------------------**

**insert into employeeTransportation values (121,'1200','Gurgaon')**

**insert into employeeTransportation values (122,'1200','Gurgaon')**

**insert into employeeTransportation values (123,'1400','Gurgaon')**

**insert into employeeTransportation values (124,'1200','Delhi'**)

**insert into employeeTransportation values (125,'1200','Faridabad')**

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

**select \* from employeeTransportation**

**o/p**

**empid allowance empaddress**

121 1200 Gurgaon

122 1200 Gurgaon

123 1400 Gurgaon

124 1200 Delhi

125 1200 Faridabad

**alter table employeetransportation add Tempid varchar(50)**

**---------------------------------------------------------------------------------------------------------------------**

**update employeeTransportation set tempid=121 where empid=121**

**update employeeTransportation set tempid=122 where empid=122**

**update employeeTransportation set tempid=123 where empid=123**

**update employeeTransportation set tempid=124 where empid=124**

**update employeeTransportation set tempid=125 where empid=125**

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

**join on three tables**

one table transportation having no foreign key but empid is same with all other two tables still it will give output by applying join but processing will be slow to

get more data if we have thousands of records so always have refrence key )

**select fullname , salary ,allowance from EmployeeDetails join EmployeSalary on EmployeeDetails.empid=EmployeSalary.empid join employeeTransportation**

**on EmployeSalary.empid= employeeTransportation.empid where(salary>12000 and allowance>1200)**

**o/p**

**fullname salary allowance**

Nik 220000.00 1400

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

**Q9)Write a SQL query to fetch all the Employees who are also managers from EmployeeDetails table.**

Ans.

Here, we have to use Self-Join as the requirement wants us to analyze the EmployeeDetails table as two different tables, each for Employee and manager records.

we have to use alias of table otherwise it will give error of ambiguity coz we use self join means join on same table

**select e.fullname from EmployeeDetails e join EmployeeDetails f on e. empid= f.ManagerId**

**select distinct e.fullname from EmployeeDetails e join EmployeeDetails f on e. empid= f.ManagerId**

distinct will help to give unique employees name if u have duplicity in table

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

# **Referential Integrity(foreign key in child table which will be primar key in parent table)**

#(without having foreign key in employeefacility table join is possible by having same coulumn name and same value as empid column

in employee details and employeefacilities but it is not the safest thing in the database as

foreign key is used to prevent update/deletion anomalies(duplicacy) in database to prevent discrepencies.Means that

nobody can not delete the data from any table coz the table having foreign key which will be primary key of another

so it will not be possible to delete the data or update the data from table having foreign key)

**for example**

1) employesalary table having empid as foreign key by employedetails's column empid as primary key now i m trying to add or insert

data in employeesalary it will not be possible coz i am adding new employeeid 130 which is not having in main table employeetable having empid as primary key

**insert EmployeSalary values (130,'p2',440000)**

**ERROR**(foreign key is empid we are giving value is 130 which is not in parent table(having empid as primary key)insertion on foreign key is not possible

Msg 547, Level 16, State 0, Line 21

The INSERT statement conflicted with the FOREIGN KEY constraint "FK\_\_EmployeSa\_\_empid\_\_03F0984C".

The conflict occurred in database "incedo", table "dbo.EmployeeDetails", column 'empid'.

The statement has been terminated.

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

**2) No want to update the foreign key in child table employesalary**

**update EmployeSalary set empid=1277 where project='p1'**

**ERROR**

Msg 547, Level 16, State 0, Line 28

The UPDATE statement conflicted with the FOREIGN KEY constraint "FK\_\_EmployeSa\_\_empid\_\_03F0984C".

The conflict occurred in database "incedo", table "dbo.EmployeeDetails", column 'empid'.

The statement has been terminated.

**So, updation or Insertion is not possible on the column of foreign key of table**

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

**(without foreign key )**

**create table employeeFacilities ( empid varchar(50), facilities varchar(100))**

**---------------------------------------------------------------------------------------------------------------------**

**insert into employeeFacilities values(121,'HeadPhones')**

**insert into employeeFacilities values(122,'Dongal')**

**------------------------------------------------------------------------------------------------------------------**

**select \* from EmployeeDetails join employeeFacilities on EmployeeDetails.empid= employeeFacilities.empid**

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

**Advantage of Referential Integrity in RDBMS and SQL**

There are several benefit of Referential Integrity in relational database and maintaining integrity of data among parent and child tables.

Here are some of the most noticed advantages of Referential Integrity in SQL:

**1)** Referential Integrity prevents inserting records with incorrect details in table. Any insert or update operation will fail if it doesn't

satisfy referential integrity rule.

**2)** If a records from parent table is deleted, referential integrity allows to delete all related records from child table using cascade-delete functionality.

**3)** Similar to second advantage if a record i.e. customer\_id of a customer is updated on parent table (Customer table) , Referential Integrity

helps to update customer\_id in child table (Order) using cascade-update.

**DELETE CASCADE**: (Cascade need to do on Child Table only on the foreign key so use "on cascade delete"only on Foreign key)When we create a foreign key using this option,

it deletes the referencing rows in the child table when the referenced row is

deleted in the parent table which has a primary key.(if we delete any thing in parent table then automatticalyy child table data is deleted)

**example---------------------**

**CREATE TABLE Countries (CountryID INT PRIMARY KEY,CountryName VARCHAR(50),CountryCode VARCHAR(3))**

**-----------------------------------------------------------------------------------------------------------**

**CREATE TABLE states ( StateID INT FOREIGN KEY(StateID) REFERENCES Countries(CountryID) ON DELETE CASCADE, StateName varchar(50),StateCode varchar(10) )**

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

**INSERT INTO Countries VALUES (1,'United States','USA')**

**INSERT INTO Countries VALUES (2,'United Kingdom','UK')**

**INSERT INTO States VALUES (1,'Texas','TX')**

**INSERT INTO States VALUES (2,'Arizona','AZ')**

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

**select \* from Countries**

**O/P**

**CountryID CountryName CountryCode**

1 United States USA

2 United Kingdom UK

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

**Select \* from states**

**O/P**

**StateID StateName StateCode**

1 Texas TX

2 Arizona AZ

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

**Now see, I will delete the data from parent table(Countries) but automatically it will be deleted from child table(States) as well**

**---------------------------------------------------------------------------------------------------------------------**

**delete Countries where CountryID=2**

**---------------------------------------------------------------------------------------------------------------------**

**select \* from Countries**

**CountryID CountryName CountryCode**

1 United States USA

**---------------------------------------------------------------------------------------------------------------------**

**Select \* from states(here see child table is automatically deleted data d=from id=2 even we have not execute delete query on the child table state)**

**StateID StateName StateCode**

1 Texas TX

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

**UPDATE CASCADE:** When we create a foreign key using UPDATE CASCADE the referencing rows are updated in the child table when the referenced row is

updated in the parent table which has a primary key.

NOTE:DIFFERENCE BETWEEN (SELECT INTO) AND (INSERT INTO SELECT)

**###### SELECT \* INTO TARGETTABLE FROM SOURCETABLE2(COPY STRUCTURE AND TABLE)**

**###### SELECT \* INTO TARGETTABLE FROM SOURCETABLE2 WHERE 1=2(COPY ONLY STRUCTURE IN TARGET TABLE)**

**###### INSERT INTO TARGETTABLE SELECT \* FROM SOURCETABLE(COPY ONLY DATA IN TARGET TABLE FROM SOURCETABLE)**

**---------------------------------------------------------------------------------------------------------------------**

**Ques.10. Write a SQL query to create a new table with data and structure copied from another table**.

Using **SELECT INTO** command-by this it will create structure of employeedetails table as well as data is also transferred from employeedetails table to new table

which is newTbale

**SELECT \* INTO newTable FROM EmployeeDetails**

**Note :**- it will copy the details of employeedetails table to newtable once by executing this query but if u delete the data f

from newtable (delete from newtable ) then all rows will be deleted from newtable so now again there is rquirement to insert data again in newtable from employeedetails

then u will try to execute same query (select \* into newtable from employeedetails) it will throw error as coz this query already executed for new table as it is copy

the structure of table as well as data now after executing delete query we just deleted the data not structure so that is why it thrw error .

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

**(insert into newtable select \* from employeedetails)**

**or**

**insert into newTable(empid,fullname,ManagerId,DateOfJoining) select empid,fullname,ManagerId,DateOfJoining from employeedetails**

it always copy the data from one table to another table(all rows copied from employedetails to newtable)

**NOTE:**

To want copy data from one table to another some column that is also possible(full name is not copied so it will have null value in target table)

**insert into newTable(empid,ManagerId,DateOfJoining) select empid,ManagerId,DateOfJoining from employeedetails**

**o/p** 13 rows are affected

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

**select \* from newtable (o/p)**

**empid FullName ManagerId DateOfJoining**

121 NULL 321 2019-12-09

122 NULL 322 2019-12-09

123 NULL 323 2019-12-09

124 NULL 324 2019-10-04

125 NULL 325 2019-01-09

126 NULL 326 2019-02-07

127 NULL 327 2019-09-09

128 NULL 328 2019-09-09

129 NULL 126 2018-12-09

321 NULL 989 2019-12-09

322 NULL 979 2019-12-09

325 NULL 321 2019-12-09

326 NULL 321 2019-12-09

**NOTE:**we can copy some rows as well from source table to target table by applying condition in where clause

**insert into newTable(empid,ManagerId,DateOfJoining) select empid,ManagerId,DateOfJoining from employeedetails where empid in ('121','122','123','124')**

**o/p -----4 rows are affected**

**empid FullName ManagerId DateOfJoining**

**121 NULL 321 2019-12-09**

**122 NULL 322 2019-12-09**

**123 NULL 323 2019-12-09**

**124 NULL 324 2019-10-04**

**---------------------------------------------------------------------------------------------------------------------**

**Ques.11. Write a SQL query to create an empty table with same structure as some other table.**

**Ans.** Using SELECT INTO command with False 'WHERE' condition - here we just want to create structure of employeedetails table but now data then we will apply

**condition which is 1=0 0r 1=2 or 2=3**

**SELECT \* INTO newTable1 FROM EmployeeDetails WHERE 1 = 0;**

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

**Q12) Write a SQL query to find NTh higehst salary from table ?**

**ANS** : find first highest salary then we use max function

**select max(salary) from employetable**

O/P: 920000

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

similarly we can find **2nd highest** salary with max function

**select max(salary) from EmployeSalary where salary<(select max(salary) from EmployeSalary)**

--------------------------------------------------------------------------------------------------------------------- O/P: 420000

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

similarly we can find **3rd highest salary** as well with max function

**select max(salary) from EmployeSalary where salary <(select max(salary) from EmployeSalary where salary<(select max(salary) from EmployeSalary))**

O/P: 320000

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

**Note**: we can calculate by this any nth number of max salary but problem is length of query is increasing and performance is reduced so need to take another

**method to find nth highest salary from table**

**SELECT MIN(salary) FROM EmployeSalary WHERE salary IN (SELECT DISTINCT TOP N salary FROM EmployeSalary ORDER BY salary DESC)**

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

**# SubQuery** in SQL is a query inside another query. Some time to get a particular information from database you may

need to fire two separate sql queries, subQuery is a way to combine or join them in single query. SQL query which

is on inner part of main query is called inner query while outer part of main query is called outer query. for example

in below sql query

**Null Statement**

**select \* from EmployeSalary where salary = null**

O/P

**empid project salary**

**means u will not get any output as u have to use IS NULL and IS NOT NULL**

**--------------------------------------------------------------------------------------------------------------------**

**select \* from EmployeSalary where salary is null**

**O/P**

**Empid project salary**

128 p3 NULL

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

**select \* from EmployeSalary where salary is not null**

**empid project salary**

121 p1 120000.00

122 p1 320000.00

123 p2 220000.00

124 p2 420000.00

125 p3 920000.00

126 p3 190000.00

127 p3 12333.00

129 p1 120000.00

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

**OFFSET FETCH**

OFFSET FETCH clauses to limit the number of rows returned by a query.

**Note** that you must use the **OFFSET and FETCH** clauses with **the ORDER BY clause**. Otherwise, **you will get an error.**

**Offset means skip that rows what u will mention rest of the rows will be the output**

**Select \* from employeedetails order by fullname asc**

**empid FullName ManagerId DateOfJoining**

322 Ander 979 2019-12-09

126 diyan 326 2019-02-07

127 herry 327 2019-09-09

121 John Perry 321 2019-12-09

325 john perry 321 2019-12-09

326 john perry 321 2019-12-09

122 keran kerry 322 2019-12-09

123 Nik 323 2019-12-09

124 Olga 324 2019-10-04

129 olga 126 2018-12-09

321 olga 989 2019-12-09

125 Pamela 325 2019-01-09

128 perry 328 2019-09-09

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

**select \* from EmployeeDetails order by FullName asc offset 9 rows**

**empid FullName ManagerId DateOfJoining**

129 olga 126 2018-12-09

321 olga 989 2019-12-09

125 Pamela 325 2019-01-09

128 perry 328 2019-09-09

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

**select \* from EmployeeDetails order by FullName asc offset 3 rows fetch next 2 rows only**

**or**

**select \* from EmployeeDetails order by FullName asc offset 3 rows fetch first 2 rows only**

**(Difference in FIRST or NEXT)use any word**

**empid FullName ManagerId DateOfJoining**

121 John Perry 321 2019-12-09

325 john perry 321 2019-12-09

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

**select \* from EmployeeDetails ~~order by FullName asc~~ offset 3 rows fetch next 2 rows only**

**Error:** Msg 102, Level 15, State 1, Line 7

Incorrect syntax near '3'.

Msg 153, Level 15, State 2, Line 7

Invalid usage of the option next in the FETCH statement.

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

--The OFFSET clause specifies the number of rows to skip before starting to return rows from the query.The offset\_row\_count

can be a constant, variable, or parameter that is greater or equal to zero.

---The FETCH clause specifies the number of rows to return after the OFFSET clause has been processed. The offset\_row\_count

can a constant, variable or scalar that is greater or equal to one.

---The OFFSET clause is mandatory while the FETCH clause is optional. Also, the FIRST and NEXT are synonyms respectively

so you can use them interchangeably. Similarly, you can use the FIRST and NEXT interchangeably.

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

**# LIKE operator:-** The SQL Server LIKE is a logical operator that determines if a character string matches a specified pattern

The LIKE operator is used in the [**WHERE**](https://www.sqlservertutorial.net/sql-server-basics/sql-server-where/) clause of **the**[**SELECT**](https://www.sqlservertutorial.net/sql-server-basics/sql-server-select/)**,**[**UPDATE**](https://www.sqlservertutorial.net/sql-server-basics/sql-server-update/)**, and**[**DELETE**](https://www.sqlservertutorial.net/sql-server-basics/sql-server-delete/)statements to filter rows based on pattern matching.

. A pattern may include regular characters and wildcard characters**(%, \_, [ ], ^)**

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

1. **% :- used by two ways**
2. %letter :- means match the pattern that having letter in last of % percentage sign

Means word or string ends with that letter

**Ex.**

select \* from EmployeeDetails where FullName **like '%a'**

**empid FullName ManagerId DateOfJoining**

124 Olg**a** 324 2019-10-04

125 Pamel**a** 325 2019-01-09

129 olg**a** 126 2018-12-09

321 olg**a** 989 2019-12-09

**(name ends with a)**

**---------------------------------------------------------------------------------------**

select \* from EmployeeDetails where FullName **like '%ga'**

**empid FullName ManagerId DateOfJoining**

124 Ol**ga** 324 2019-10-04

129 ol**ga** 126 2018-12-09

321 ol**ga** 989 2019-12-09

**(name ends with ga)**

**---------------------------------------------------------------------------------------**

1. **letter %:-** means match the pattern that having letter in before of % percentage sign(means string or word start with that letter)

select \* from EmployeeDetails where FullName **like 'j%'**

**empid FullName ManagerId DateOfJoining**

121 **J**ohn Perry 321 2019-12-09

325 **j**ohn perry 321 2019-12-09

326 **j**ohn perry 321 2019-12-09

**(Name starts with J)**

**---------------------------------------------------------------------**

select \* from EmployeeDetails where FullName **like 'o%a'**

**empid FullName ManagerId DateOfJoining**

124 **O**lg**a** 324 2019-10-04

129 **o**lg**a** 126 2018-12-09

321 **o**lg**a** 989 2019-12-09

**(name starts with O and ends with A**

**----------------------------------------------------------------------**

**2. [] it is used for multiple options like starts with J and D both**

select \* from EmployeeDetails where FullName **like '[jd]%'**

**empid FullName ManagerId DateOfJoining**

121 **J**ohn Perry 321 2019-12-09

126 **d**iyan 326 2019-02-07

325 **j**ohn perry 321 2019-12-09

326 **j**ohn perry 321 2019-12-09

**(Name Starts with J and D )**

**----------------------------------------------------------------------**

select \* from EmployeeDetails where FullName **like '%[yn]'**

**empid FullName ManagerId DateOfJoining**

121 John Perr**y** 321 2019-12-09

122 keran kerr**y** 322 2019-12-09

126 diya**n** 326 2019-02-07

127 herr**y** 327 2019-09-09

128 perr**y** 328 2019-09-09

325 john perr**y** 321 2019-12-09

326 john perr**y** 321 2019-12-09

**(name ends with Y and N both )**

**----------------------------------------------------------------------**

**[J-O] Means letters which comes between J-O which are J,K,L,M,N,O names start with these letters**

select \* from EmployeeDetails where FullName **like '[j-o]%'**

**empid FullName ManagerId DateOfJoining**

121 **J**ohn Perry 321 2019-12-09

122 **k**eran kerry 322 2019-12-09

123 **N**ik 323 2019-12-09

124 **O**lga 324 2019-10-04

129 **o**lga 126 2018-12-09

321 **o**lga 989 2019-12-09

325 **j**ohn perry 321 2019-12-09

326 **j**ohn perry 321 2019-12-09

**(Names start with j,k,l,m,n)**

**----------------------------------------------------------------------**

**3 ) ^ or NOT(except the condition u will be given)**

select \* from EmployeeDetails where FullName **not like '[j-o]%'**

**or**

select \* from EmployeeDetails where FullName **like '[^j-o]%'**

**empid FullName ManagerId DateOfJoining**

125 **P**amela 325 2019-01-09

126 **d**iyan 326 2019-02-07

127 **h**erry 327 2019-09-09

128 **p**erry 328 2019-09-09

322 **A**nder 979 2019-12-09

**(Names start without starting j,k,l,m,n)**

**-----------------------------------------------------------------**

**4. Underscore (\_) :- one underscore means left one letter**

**Two underscore means left two letter**

select \* from EmployeeDetails where FullName **like '\_a%'**

empid FullName ManagerId DateOfJoining

125 P**a**mela 325 2019-01-09

**(names having 2nd letter is a as left one space)**

**----------------------------------------------------------------------**

select \* from EmployeeDetails where FullName **like '\_\_r%'**

empid FullName ManagerId DateOfJoining

122 ke**r**an kerry 322 2019-12-09

127 he**r**ry 327 2019-09-09

128 pe**r**ry 328 2019-09-09

**(names having 3rd letter is r as left two spaces)**

**---------------------------------------------------------------------------------------------------------------------**

**BETWEEN:**- this operator used only with Numbers and Dates, it will give data which lies between and condition

select \* from EmployeeDetails where empid **between 123 and 126**

**empid FullName ManagerId DateOfJoining**

**123** Nik 323 2019-12-09

**124** Olga 324 2019-10-04

**125** Pamela 325 2019-01-09

**126** diyan 326 2019-02-07

**(Got details only between 123 to 126 which are 123,124,125,126)**

**NOT BETWEEN:**

select \* from EmployeeDetails where empid **not between 121 and 129**

empid FullName ManagerId DateOfJoining

321 olga 989 2019-12-09

322 Ander 979 2019-12-09

325 john perry 321 2019-12-09

326 john perry 321 2019-12-09

**SELECT DISTINCT**

 it removes the duplicate values in the column from the result set.

**NOTE**:

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.

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

select distinct FullName from EmployeeDetails



**NOTE:**

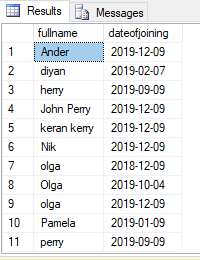
Both DISTINCT and GROUP BY clause reduces the number of returned rows in the result set by removing the duplicates.

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.**

select distinct FullName, DateOfJoining from EmployeeDetails

or

select fullname ,dateofjoining from EmployeeDetails group by FullName,DateOfJoining

**(Same output for same input by using distinct and group by)**

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

**Data Variable:**

variables are the object which acts as a placeholder to a memory location. Variable hold single data value.

**It is of two types**

1. **Local Variable**
2. **Global Variable(**The system maintains the global variable**.**A user cannot declare them. The global variable starts with **@@.** It stores **session related information**.

**Local Variable:**

1. A user declares the local variable.
2. By default, a local variable starts with **@.**
3. Every local variable scope has the restriction to the **current batch or procedure** within any given session.

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

**#Syntax to declare variable withoutr assigning any value**

**DECLARE @empid INT;**

**----------------------------------------------------------------------**

**#Syntax to cdeclare variable and assigning value .as well print that value but condition is that we cant print this variable separately ,it will execute with declare statement collectively**

**Declare @empid int=5**

**Print @empid**

**o/P 5**

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

**print @empid**

**(if u print single statement to print value of declared variable then it will throw error)**

**o/p**

M**sg 137, Level 15, State 2, Line 18**

**Must declare the scalar variable "@course\_ID".**

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

**Declare multiple variable and assign its values(by using keyword SET) and print it (by using print or select keyword)**

declare @empid int , @name varchar(50) , @course varchar(50)

**set** @empid=5

**set** @name='veera'

**set** @course='btech'

print @empid

print @name

print @course

**o/p**

**5**

**veera**

**btech**

**Assigning a value to a variable with a Scalar Subquery using SET**

**Note** A scalar query is a query with results as just one row and one column. Otherwise, the query will throw an error.

**declare @name varchar(50)**

**set @name= (select fullname from employeedetails where empid =123)**

**print @name**

**o/p NIK**

**---------------------------------------------------------------------------------------------------------------------**

**(instead of using print we can use select by this it gives output in table by any column name if u mention the name of column like (AS name)**

declare @@name varchar(50)

set @@name= (select fullname from employeedetails where empid =123)

**select @@name as name**



**NOTE:Simple variable always return a single value like we are selecting maximum empid from employeedetails table**

**Query:**

**1)**

**declare @id int=(select max(empid) from EmployeeDetails)**

**print @id**

**O/P 325**

but we want to get **multiple values** in a **single variable** like distinct all empid from employeedetails table, it will throw error

**Query:**

**2)**

**declare @id int=(select distinct empid from EmployeeDetails)**

**print @id**

**Msg 512, Level 16, State 1, Line 28**

**Subquery returned more than 1 value. This is not permitted when the subquery follows =, !=, <, <= , >, >= or when the subquery is used as an expression.**

**Even we can’t store multiple column values in a single variable it will give error**

**Query**

**3)**

**declare @id int=(select empid,FullName from EmployeeDetails)**

**print @id**

**Msg 116, Level 16, State 1, Line 29**

**Only one expression can be specified in the select list when the subquery is not introduced with EXISTS.**

**TABLE VARIABLE:- In a simple variable we can store a single value if we want to store multiple values in a single variable and multiple columns in a single variable then we have to use a table variable.**

declare @id table(eid varchar(50), name varchar(50))

insert into @id values('121','seema')

insert into @id values('122','reema')

select \* from @id

**o/p**

**eid name**

121 seema

122 reema

**Values simply inserted through insert statement not by set command directly by insert command**

**data is inserted into table variable throw fetching from other table**

declare @id table(eid varchar(50), name varchar(50))

insert into @id select empid,fullname from EmployeeDetails

select eid ,name from @id

**execute these statement collectively**

**o/p**

**eid name**

121 John Perry

122 keran kerry

123 Nik

124 Olga

125 Pamela

126 diyan

127 herry

128 perry

129 olga

321 olga

325 john perry

So means that we can store multiple column as well as multiple rows in a single variable but it should be a table variable

declare @id table(eid varchar(50), name varchar(50))

go give error

insert into @id select empid,fullname from EmployeeDetails

go give error

select eid ,name from @id

**go give error**

**give error when we execute all commands separately so need to execute collectively, then it will be executed successfully**

**Temporary Table:- temporary table always store in temporary database it is of two types**

**1 local temp table**

**2 global temp table**

1. **Local Temp table: it is represented by single pound symbol#**

**Its output will be remains in the same session not in other means if u open new window in sql then u want to use values of local temp table then it will give error**

create table #temp(empid int,name varchar(50))

go

Command(s) completed successfully.

insert into #temp values(1,'Shyna')

insert into #temp values(2,'Myna')

go

(1 row(s) affected)

(1 row(s) affected)

select \* from #temp

go

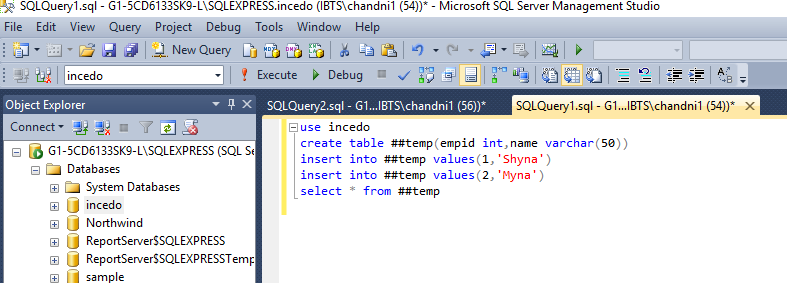
**empid name**

**1 Shyna**

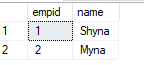
**2 Myna**

**Means in temporary table we can execute single statement but not in table variable**

**2)Global Temporary Table:- it is represented by double pound symbol## .it can also be used in other session as well.**

****

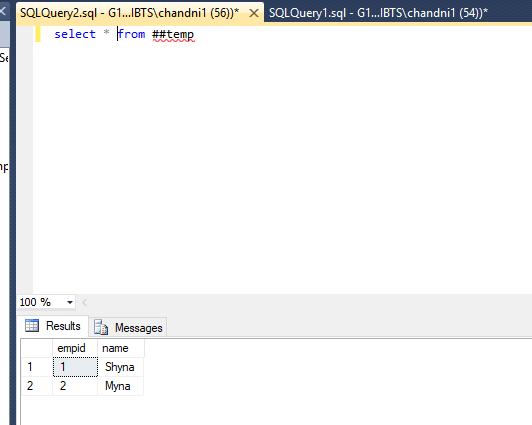
**o/p**

****

**See in this we have created global temporary table with same name as local table but difference is double pound symbol**

**Benefit of global temp table:**

**We can use in another session as well means we will open another window it will also show output**

****

**See here chandni56 window we use global temp table but created in chandni1(54) still it is giving output**

**Difference between Table Variable and Temporary table**

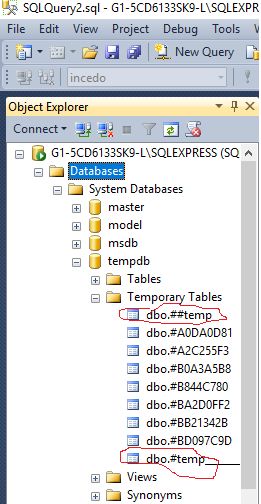
**1Table variable can’t be used in transaction and locking however temp table can be used in both**

**2 Table variable can’t be altered but temp table can be altered**

**3 Table variable can’t be truncated but temp table can be truncated**

**4 Index can’t be added into Table Variable ,but primary constraint and unique constraint can be added in table variable however in temp table can any constraint and indexes**

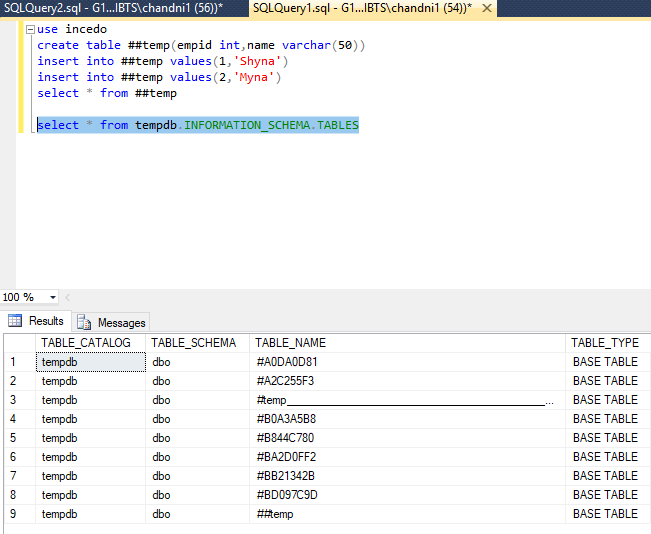
**5 Access of table variable is faster because it is stored in system memory as RAM so that is why we can’t find this in sql server Management(SSM) however temp table is stored in temp database acces of temp table is slower as it is found in temp db in SSM**

****

**or**

**By this following query we can find temp tables]**

**Select \* from tempdb.INFORMATION\_SCHEMA.TABLES**

****

**---------------------------------------------------------------------------------------**

**Difference Between Corelated Query and Non correlated query**

1.In case of **correlated subquery** inner query depends on outer query while in case of noncorrelated query inner query or subquery doesn't depends on outer query and run by its own.

2.In case of correlated subquery, outer query executed before inner query or subquery while in case of NonCorrelated subquery inner query executes before outer query.

3.Correlated Sub-queries are slower than non correlated subquery and should be avoided in favor of sql joins.

4.Common example of correlated subquery is using exits and not exists keyword while non correlated query mostly use IN or NOT IN keywords.

## SubQuery vs Join in SQL

Any information which you retrieve from database using subquery can be retrieved by using different types os joins also. Since SQL is flexible and it provides different way of doing same thing. Some people find SQL Joins confusing and subquery specially noncorrelated more intuitive but in terms of performance SQL Joins are more efficient than subqueries.

**Important points about SubQuery in DBMS**

1.Almost whatever you want to do with subquery can also be done using join, it just matter of choice

subquery seems more intuitive to many user.

2.Subquery normally return an scaler value as result or result from one column if used along with

IN Clause.

3.You can use subqueries in four places: subquery as a column in select clause,

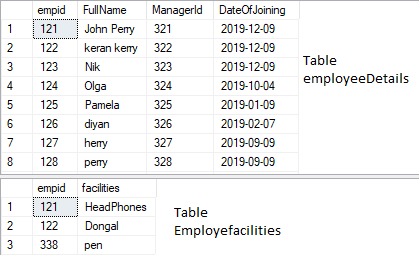
4.In case of correlated subquery outer query gets processed before inner query.

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

select \* from EmployeeDetails

select \* from employeeFacilities

**execute**



**JOIN**

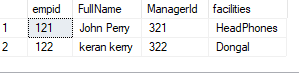
**using join but in this case we can get name of facility as well (is good to use coz it make cartesian product of two tables in a buffer or temporary table so easy to filter out**

**required data but need more space but nowadays to get sapce is easy task by cloud in case of subquery whether it is corelated or non corelated it needs comparison so more**

**comparison means need more input output operations so it consumes more time.**

select a.empid ,a.FullName ,a.ManagerId,b.facilities from EmployeeDetails a inner join employeeFacilities b on a.empid=b.empid-- and facilities in('dongal', 'headphones')

O/P



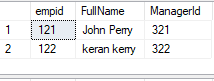
**Co-Related Query or Nested Query(it mostly use exists or not exists)**

**using exists and co related query by this we can’t get facility column of another table in this co related query firstly outer query execute then result of outer query compare data with inner query (inner query executes multiple times, means until its last record of table) that is why called nested query then inner so after first row comparison then gain takes second row from outer query then again do comparison with inner query multiples times so do it like same**

**Nested Query :-( like for (i=1; i<3 )( j=1;j<2) outer loop execute for i=1 and j will execute for two times j=1 and j=2 as similar correlated query execute for first row of outer query record and then for multiple rows for inner query)**

select a.empid ,a.FullName ,a.ManagerId from EmployeeDetails a where **exists**  (select facilities from employeeFacilities b where a.empid= b.empid )

O/P



**Exists:**

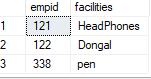
it always give response True or False if inner query gives any matching result with respect to outer loop then return true then will print the result of outer loop

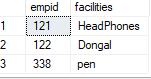
select a.empid ,a.FullName ,a.ManagerId from EmployeeDetails a where **exists**  (select facilities from employeeFacilities b where a.empid= b.empid )

here first outer loop select first record from employeedetails which is

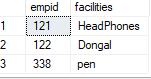


Now it will compare with inner loop multiple times until it matches with inner loop record

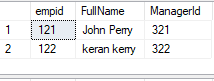
 so it get 121 record here EXISTS give response **TRUE** so now it will print 121 record and store in buffer then execute outer loop 122 record and again do comparison for all inner records record

  make comparison

So get 122 record and so EXISTS give resonse again **TRUE** so store result of 122 in buffer now go to outer loop for 3rd row record which is

  there is no comparison so EXISTS give response **FALSE** so outer loop will not print any Record

So this is the way EXISTS work and CO-RELATED Query and the final output is



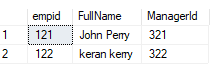
**Non Co-Related Query(it mostly use in or not in)**

**using simple non co related subquery but here also we will not get facility column (in this firstly inner query excute only once then execute outer query means after getting inner query**

**--result ,result will be compared with outer query rows then give ans)**

select a.empid ,a.FullName ,a.ManagerId from EmployeeDetails a where empid in (select empid from employeeFacilities b where a.empid=b.empid

O/P

****

--Merge : it is basically used for merging two tables but condition both have same columns with same datatype not particulary name .Purpose of merging is source

-- table have data but not in target table so need to insert data in target table and sometimes target table have extra information comparitive

-- to source table so need tol delete information in target table with respect to source table apart from that, sometimes information is not

-- matched in target table with respect to source table so need to update target table all in all we can say that at a same time all the

--operations insert ,update and delete are done at same time with the help of Merge Query

CREATE TABLE category (

category\_id INT PRIMARY KEY,

category\_name VARCHAR(255) NOT NULL,

amount DECIMAL(10 , 2 )

);

INSERT INTO category(category\_id, category\_name, amount)

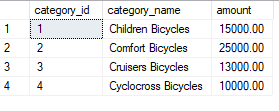
VALUES(1,'Children Bicycles',15000),

(2,'Comfort Bicycles',25000),

(3,'Cruisers Bicycles',13000),

(4,'Cyclocross Bicycles',10000);

select \* from category



CREATE TABLE category\_staging (

category\_id INT PRIMARY KEY,

category\_name VARCHAR(255) NOT NULL,

amountt DECIMAL(10 , 2 )

);

INSERT INTO category\_staging(category\_id, category\_name, amountt)

VALUES(1,'Children Bicycles',15000),

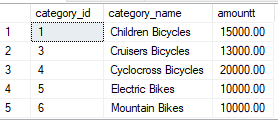
(3,'Cruisers Bicycles',13000),

(4,'Cyclocross Bicycles',20000),

(5,'Electric Bikes',10000),

(6,'Mountain Bikes',10000);

select \* from category\_staging



MERGE category t

USING category\_staging s

ON (s.category\_id = t.category\_id)

WHEN MATCHED

THEN UPDATE SET

t.category\_name = s.category\_name,

t.amount = s.amountt

WHEN NOT MATCHED BY TARGET

THEN INSERT (category\_id, category\_name, amount)

VALUES (s.category\_id, s.category\_name, s.amountt)

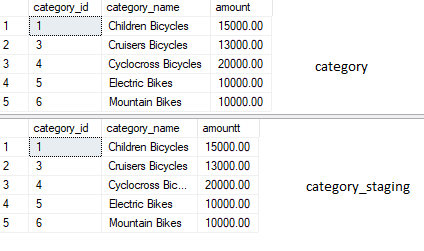
WHEN NOT MATCHED BY SOURCE

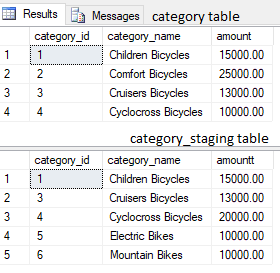
THEN DELETE;

5 rows affected

select \* from category

select \* from category\_staging



****