Database #:

Definition: A collection of related data

Database Management System (DBMS): software package / system to facilitate the creation and militance of computerized database.

Database System: refers to a system that manages databases, including the software, hardware, and procedures involved in storing, retrieving, and manipulating data typically includes the **Database Management System (DBMS)**,

Entity Relationship Diagram (ERD)#:

Definition: identifies information required by the business by displaying the relevant entities and the relationships between them

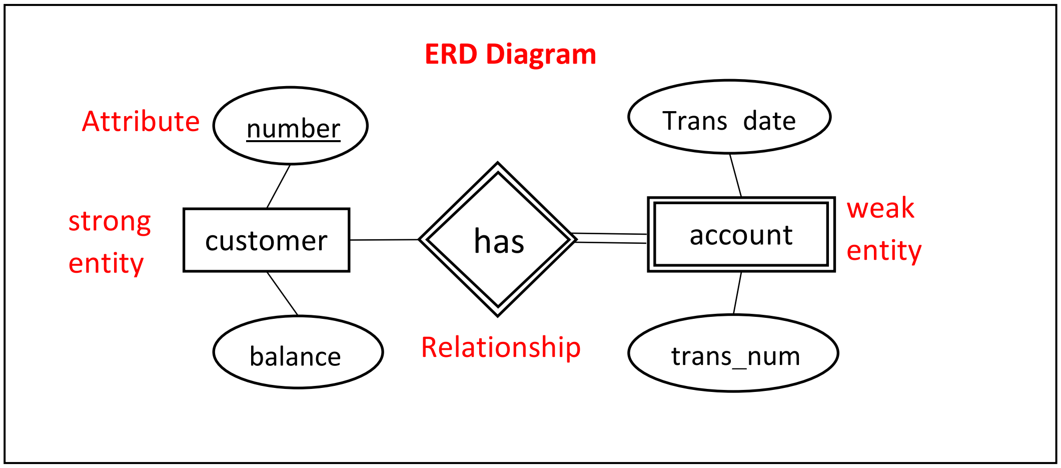
Entity #:

Definition: refers to a "thing" or "object" in a database that is distinguishable from others and typically has properties or attributes. For example, a **student** could be an entity, with attributes like **name**, **ID number**, and **age**.

Types:

1-strong Entity: **sufficient** to uniquely identify each instance of that entity. A strong entity **does not depend** on any other entity for its identification and exists independently

2- weak Entity: entity set that does not have sufficient attributes to form a **primary key** on its own. Therefore, it depends on a **strong entity** to create a unique identifier.

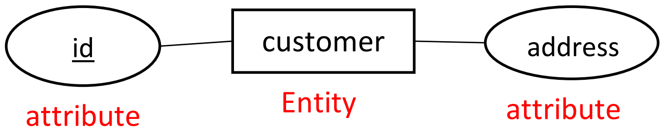


Attributes #:

Definition: the properties or fields that define an entity

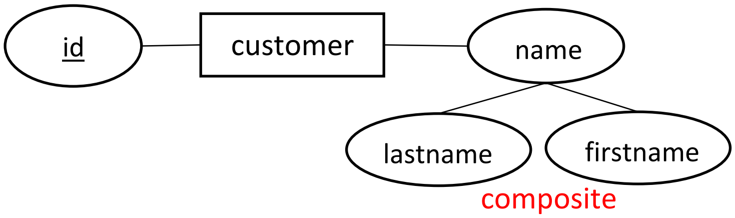
Types:

1-simple attributes:

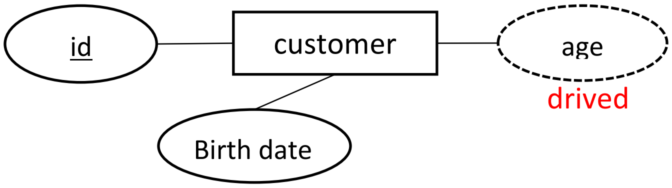


2- composite attributes:

name = firstname + lastname

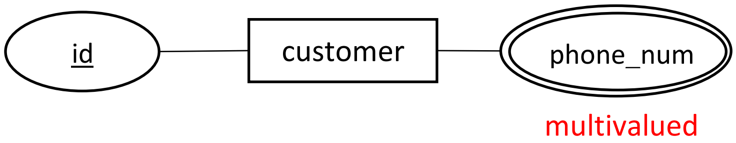


3-drived attributes: it's calculated in run time



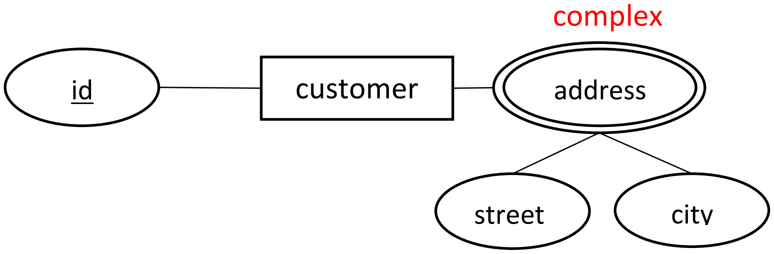
Ex: age = Today's date – birth date

4-multivalued attributes:



Ex (customer may have multiple phone number)

5-complix attributes: Composite + multivalued Ex (customer may have more address and address is composite from city and street



Relationships #:

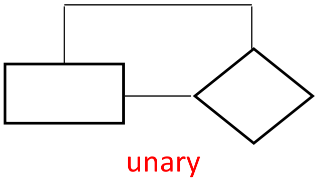
Definition: is an association among several entities

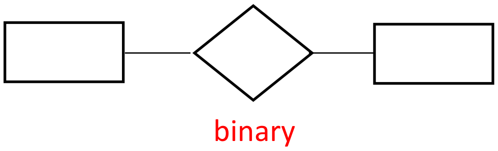
Types (Degree) of relationships:

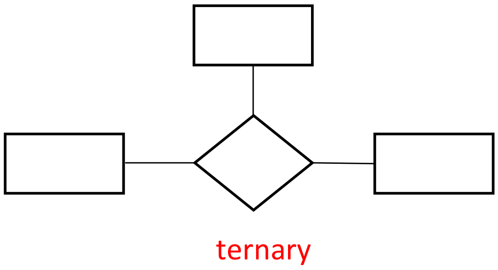
1- unary: between two instances of one entity types Ex:(person is married to person)

2- binary: between two instances of two entity types Ex:(employees work in department)

3- Ternary: among the instances of three entity types Ex:(hospital and penitent and medicine have shared relationship may be time)







Cardinality of Relationship: how many instances of one entity will or must be connected to a single instance from the other entities

1- One-to-One Relationship: each record in one table is associated with exactly one record in another table Ex:(manger mange department and department have only one manger)

2- One-to-Many Relationship: one record in the first table can be associated with many records in the second table, but each record in the second table is associated with only one record in the first table Ex:(employee work in only one department and each department has many employees)

3- Many-to-Many Relationship: Many records in the first table can be associated with Many records in the second table and vice versa Ex:(worker work in Many projects and project have many workers).

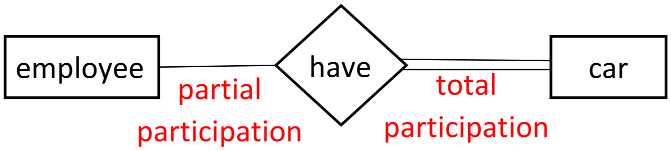
Participation Constrain #:

Definition: defines whether all entities in a particular entity set are required to participate in a relationship set.

Types:

1-Total Participation: Every entity in the entity set must participate in the relationship (cannot be null)

2-Partial Participation: Not every entity in the entity set is required to participate in the relationship (can be null)



Ex (any car in the company must be allocated to the employee but not all employees do not have a car)

Case 1

A big company has decided to store information about its projects and employees in a database. The company. Prepare an ERD diagram for this Company according to the following Description:

1- The company has a number of employees each employee has SSN, Birth Date, Gender and Name which represented as Fname and Lname.

2- The company has a set of departments each department has a set of attributes DName, DNUM (unique) and locations.

3- Employees work in several projects each project has Pname, PNumber as an identifier,Location City.

4- Each employee may have a set of dependent; each dependent has Dependent Name (unique), Gender and Birthdate. Note: if the employee left the company no needs to store his dependents info

5- For each Department, there is always one employee assigned to manage that Department and each manager has a hiring Date

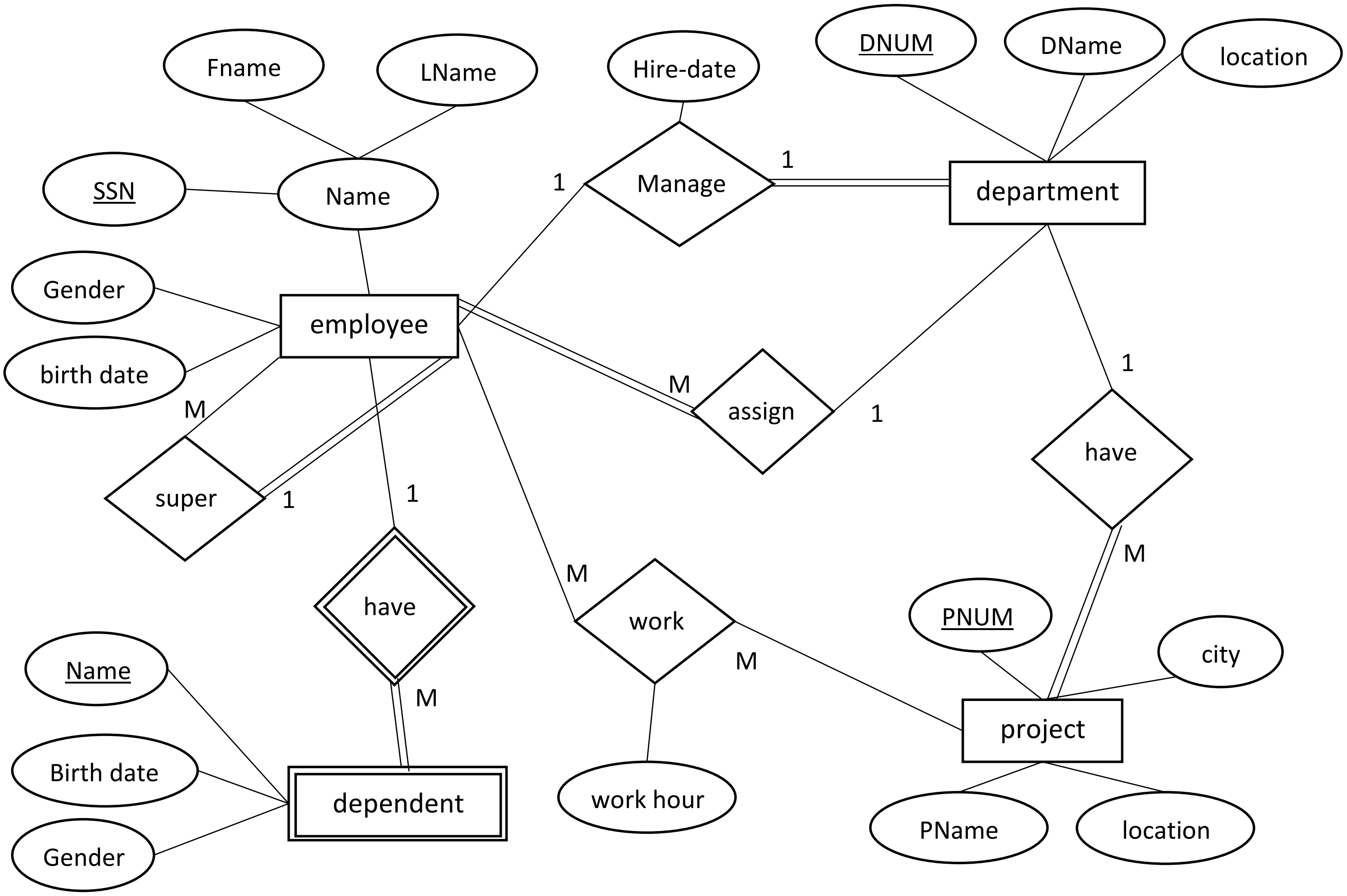
6- Department may have employees but employee must work on Only One department

7- Each department may have a set of projects and each project must assigned to one department

8- Employees work in several projects and each project has several employees and each employee has a number of working hours in each project

9- Each employee has a supervisor

Solve



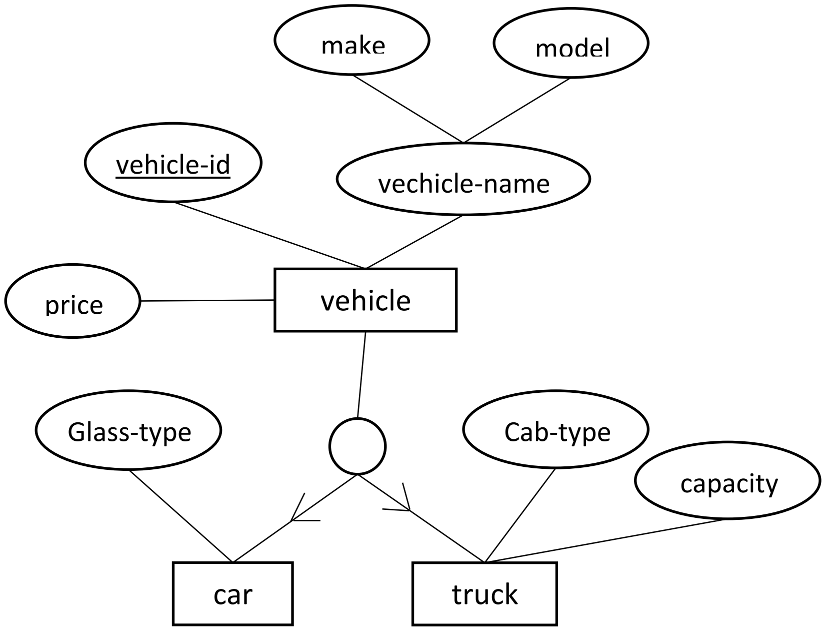
Enhanced Entity-Relationship Diagram(EERD) #:

Definition: is an advanced form of an ERD used to design and model. EERD incorporates additional concepts such as inheritance, specialization, generalization, and categories.

Types:

Subtype: A subgrouping of the entities in an entity in an entity type which has attributes that are distinct from those in other subgrouping.

Supertype: A generic entity type that has a relationship with one or more supertypes.



Constrain in Supertype:

1-Completeness Constraints:

A-Total Specialization (double line): Every instance of the supertype **must** belong to at least one subtype. Ex: Every Employee is either a manager or a technician.

B-Partial Specialization: An instance of the supertype may or may not belong to any subtype

Ex: A Vehicle may be specialized as a Car or Truck, but it can also remain a general Vehicle

2- Disjoint Constraints

A-Total Disjoint(d): An instance of supertype can belong to only one subtype.

Ex: A Vehicle can be either a Car or a Truck, but not both.

B-Overlap Rule(o): An instance of supertype can belong to multiple subtypes.

Ex: A Person can be both a Student and an Employee.

Mapping #:

Mohamed_Magdy

Definition: process of associating or linking data from one system or structure to another

Primary key: column or set of columns in a database table that uniquely identifies each row in that table (value can't repeated and can't be null)

Foreign Key: is a column (or set of columns) in one table that is used to establish a link or relationship between data in two tables. The foreign key points to a primary key or a unique key in another table.

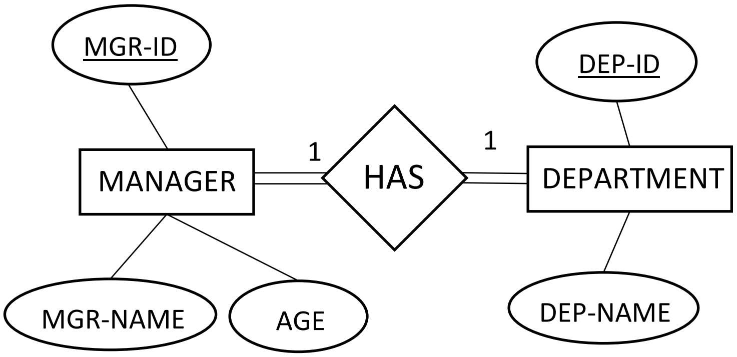
Datebase ERD Mapping #:

Definition: This term refers to the process of converting an ERB schema into database tables (or entities). This allows developers to work with objects in their code while seamlessly interacting with the relational database.

Steps: (Best Practice)

--Spatial case: First Mapping Relationship 1:1 total participation from both sides

total from both sides



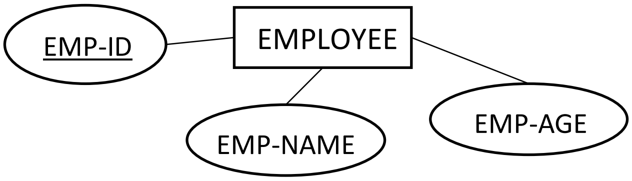
MANAGER-DEPARTEMNT

MGR-ID	MAR-NAME	AGE	DEP-ID	DEP-NAME

Primary key

1- Mapping of Regular (Strong) Entity Type

① Simple

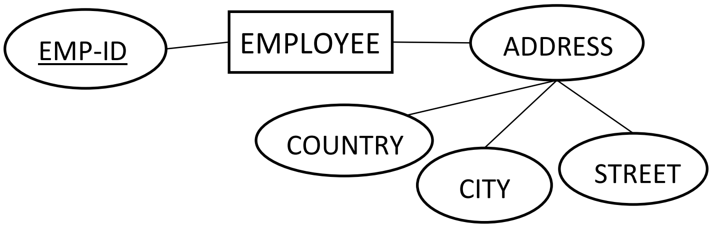


EMPLOYEE

EMP-ID	EMP_NAME	EMP-AGE

Primary key

② Composite

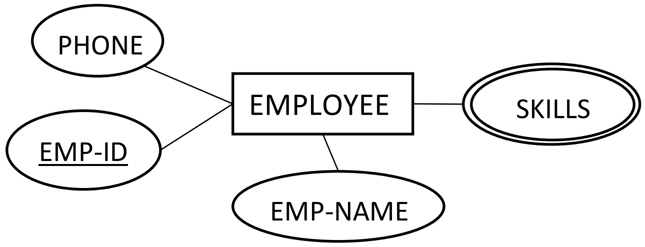


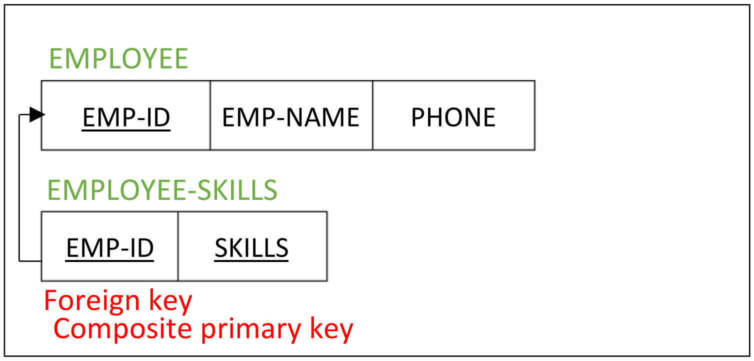
EMPLOYEE ADDRESS

EMP-ID	COUNTRY	CITY	STREET

Primary key

3 Multivalued




EMPLOYEE

EMP-ID	EMP-NAME	PHONE

EMPLOYEE-SKILLS

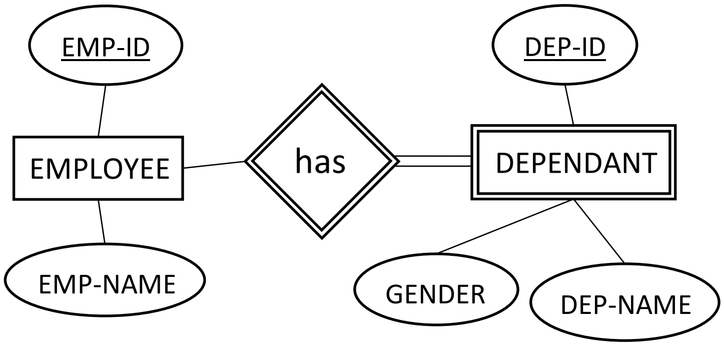
EMP-ID	SKILLS

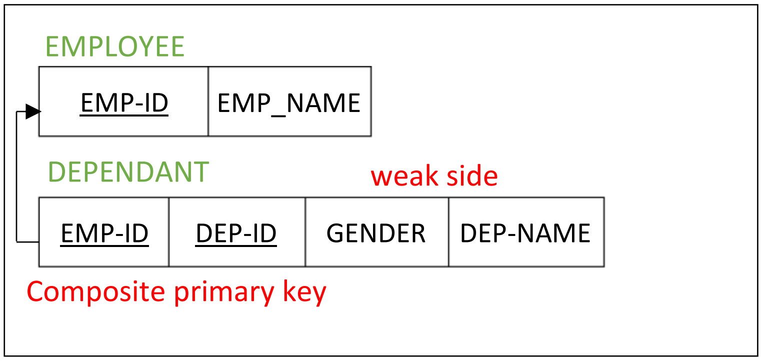
Foreign key
Composite primary key

Complex→ composite table + multivalued table

Drived → may be calculated in run time and type equation in constraint file

2- Mapping of Weak Entity Types




EMPLOYEE

EMP-ID	EMP_NAME

DEPENDANT

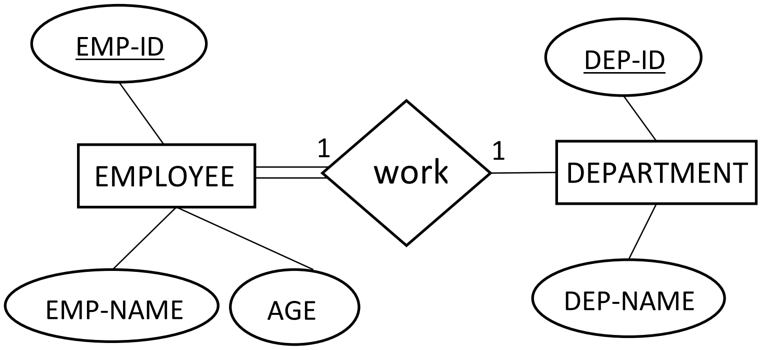
weak side

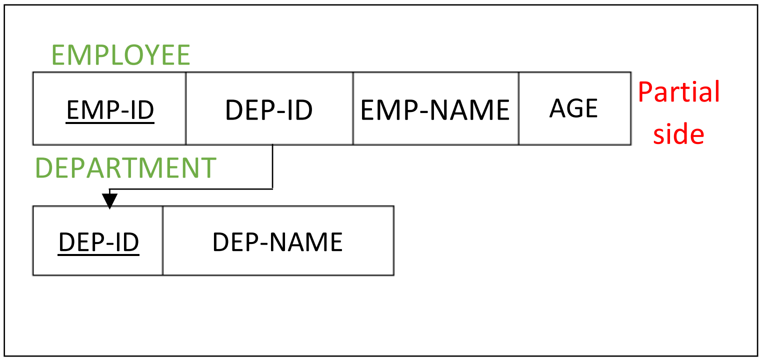
EMP-ID	DEP-ID	GENDER	DEP-NAME

Composite primary key

3- Mapping of Binary 1:1 Relationship Types

Total from side and partial from another side




EMPLOYEE

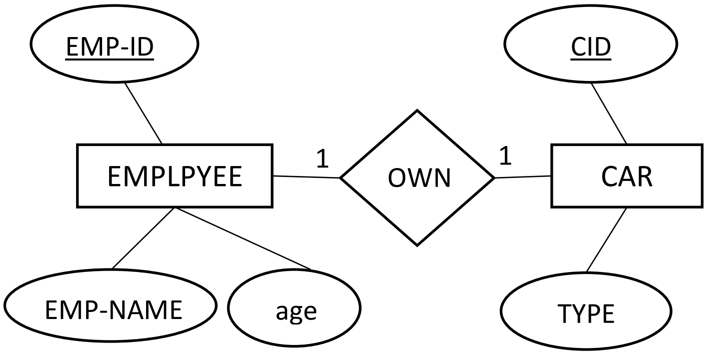
EMP-ID	DEP-ID	EMP-NAME	AGE

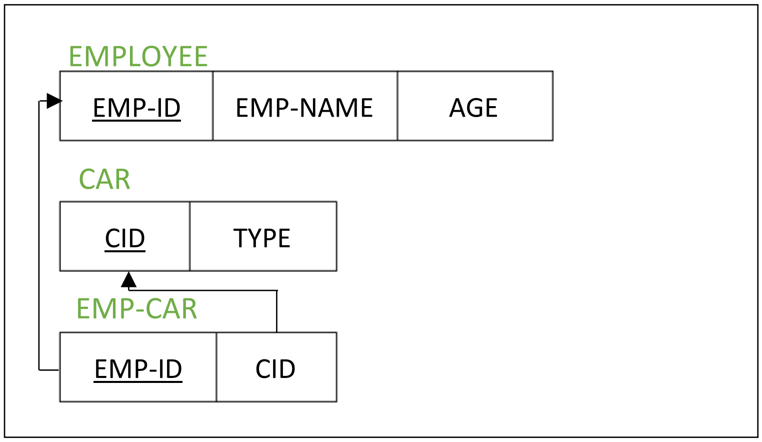
Partial side

DEPARTMENT

DEP-ID	DEP-NAME

partial from both side




EMPLOYEE

EMP-ID	EMP-NAME	AGE

CAR

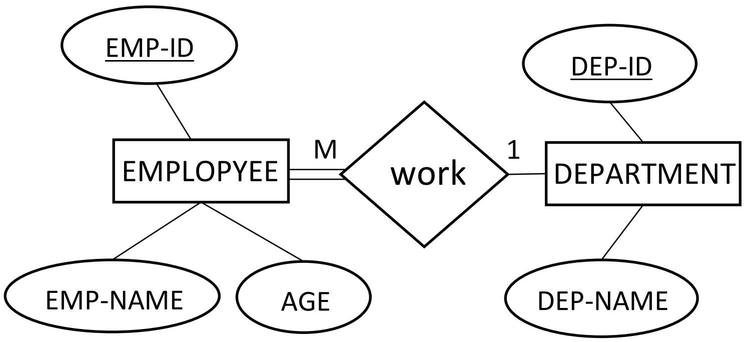
CID	TYPE

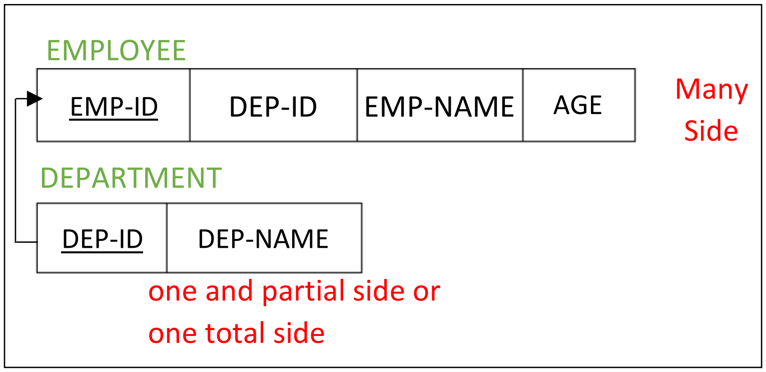
EMP-CAR

EMP-ID	CID

4- Mapping of Binary 1:M Relationship Types

Many from total side and one from other side (total or partial)





EMPLOYEE

EMP-ID	DEP-ID	EMP-NAME	AGE

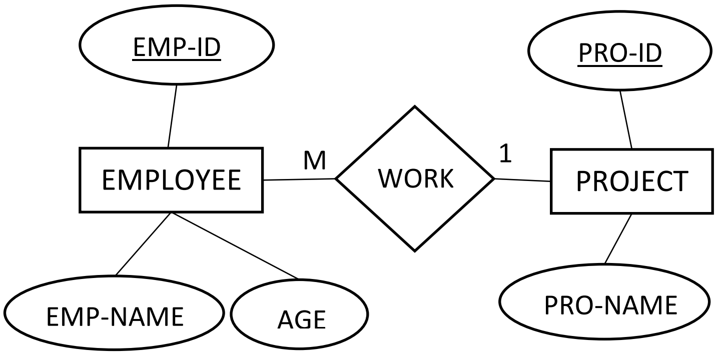
Many Side

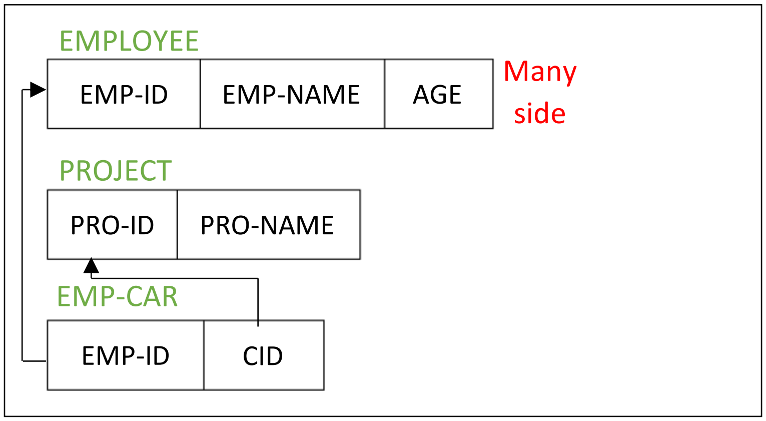
DEPARTMENT

DEP-ID	DEP-NAME

one and partial side or one total side

Many from partial side and one from total side (total or partial)





EMPLOYEE

EMP-ID	EMP-NAME	AGE

Many side

PROJECT

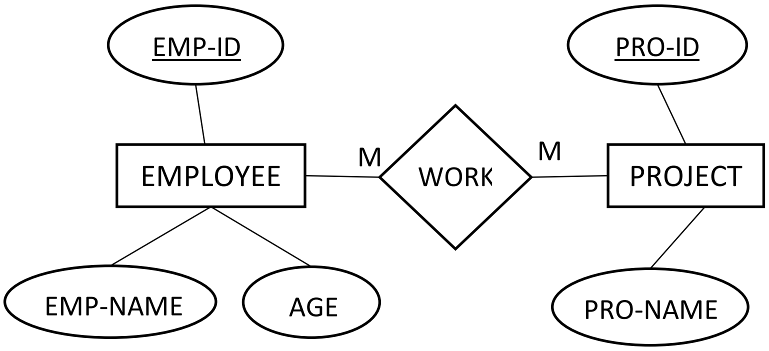
PRO-ID	PRO-NAME

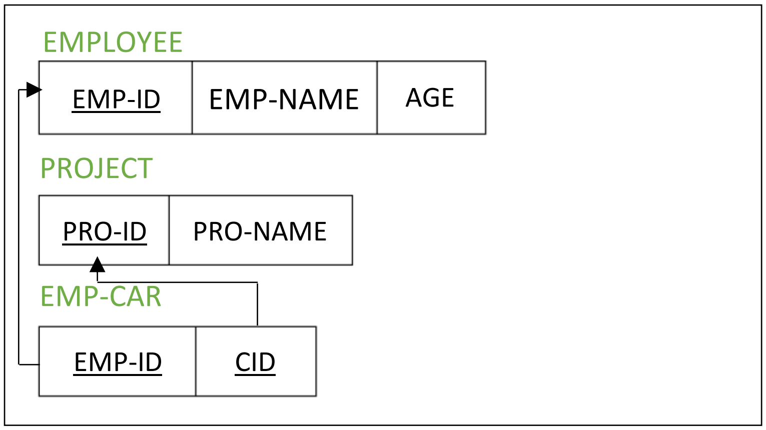
EMP-CAR

EMP-ID	CID

5- Mapping of Binary M:M Relationship Types

In all case if partial or total from any sides





EMPLOYEE

EMP-ID	EMP-NAME	AGE

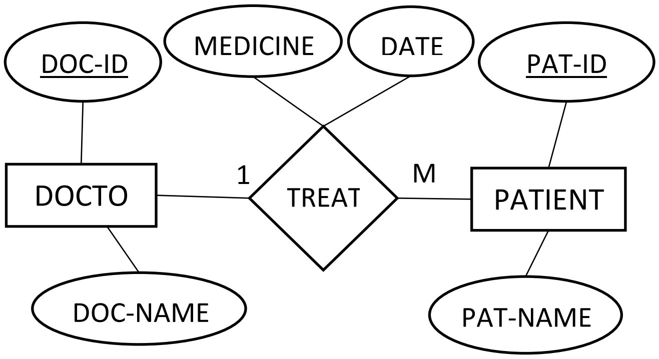
PROJECT

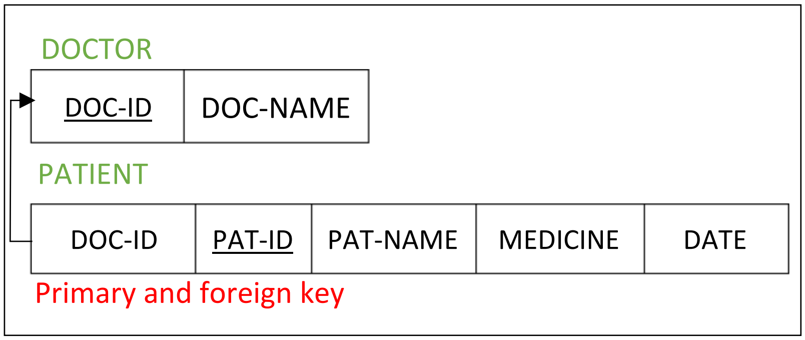
PRO-ID	PRO-NAME

EMP-CAR

EMP-ID	CID

5- Mapping of Binary M:N with attributes in Relationship : any table that appear both (primary key and foreign key together should put shared attributes on it





DOCTOR

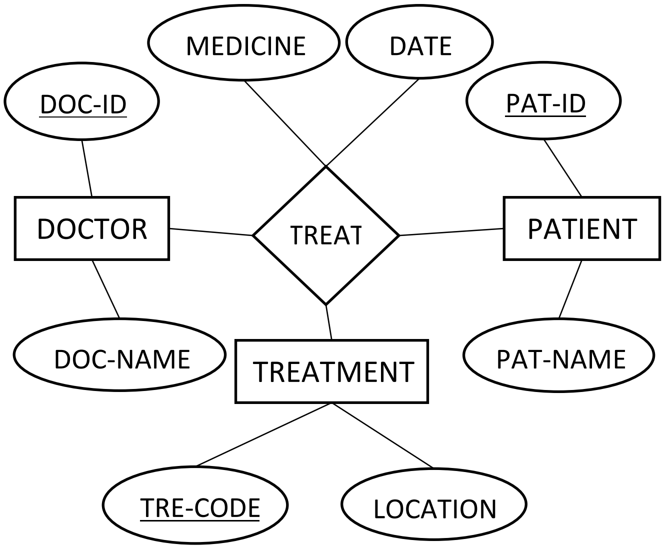
DOC-ID	DOC-NAME

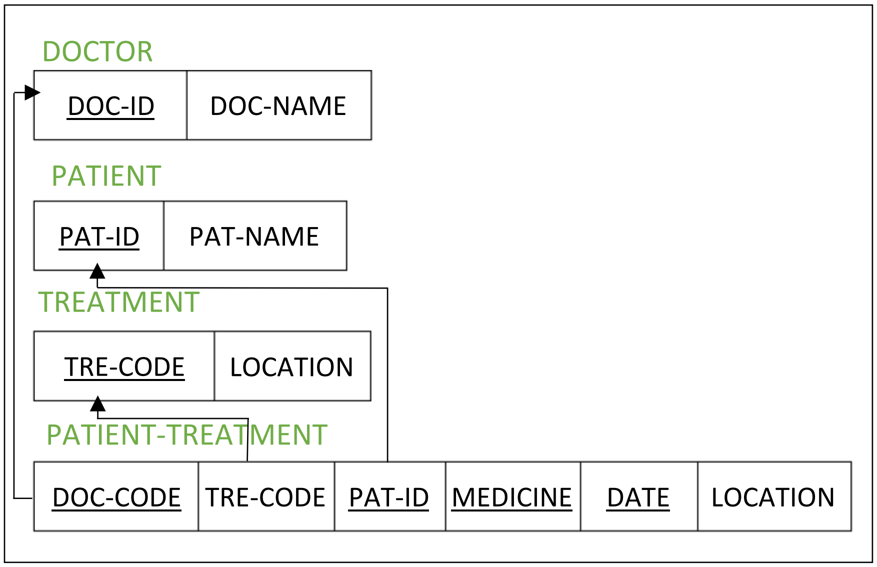
PATIENT

DOC-ID	PAT-ID	PAT-NAME	MEDICINE	DATE

Primary and foreign key

6- Mapping of N-ary(Ternary) Relationship Types: after creating all tables between relation individual and create shared tabled put all primary key on it as foreign key and search of a new composite primary key on shared table





DOCTOR

DOC-ID	DOC-NAME

PATIENT

PAT-ID	PAT-NAME

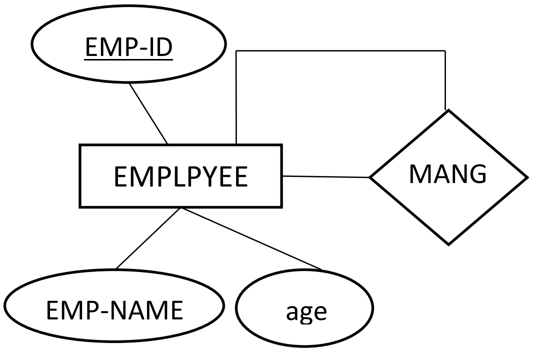
TREATMENT

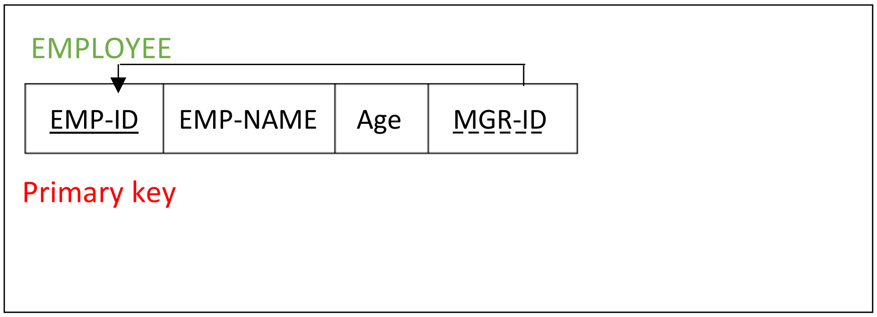
TRE-CODE	LOCATION

PATIENT-TREATMENT

DOC-CODE	TRE-CODE	PAT-ID	MEDICINE	DATE	LOCATION

7- Mapping of Unary Relationship Types: put column represent relationship in same table foreign key point in primary key



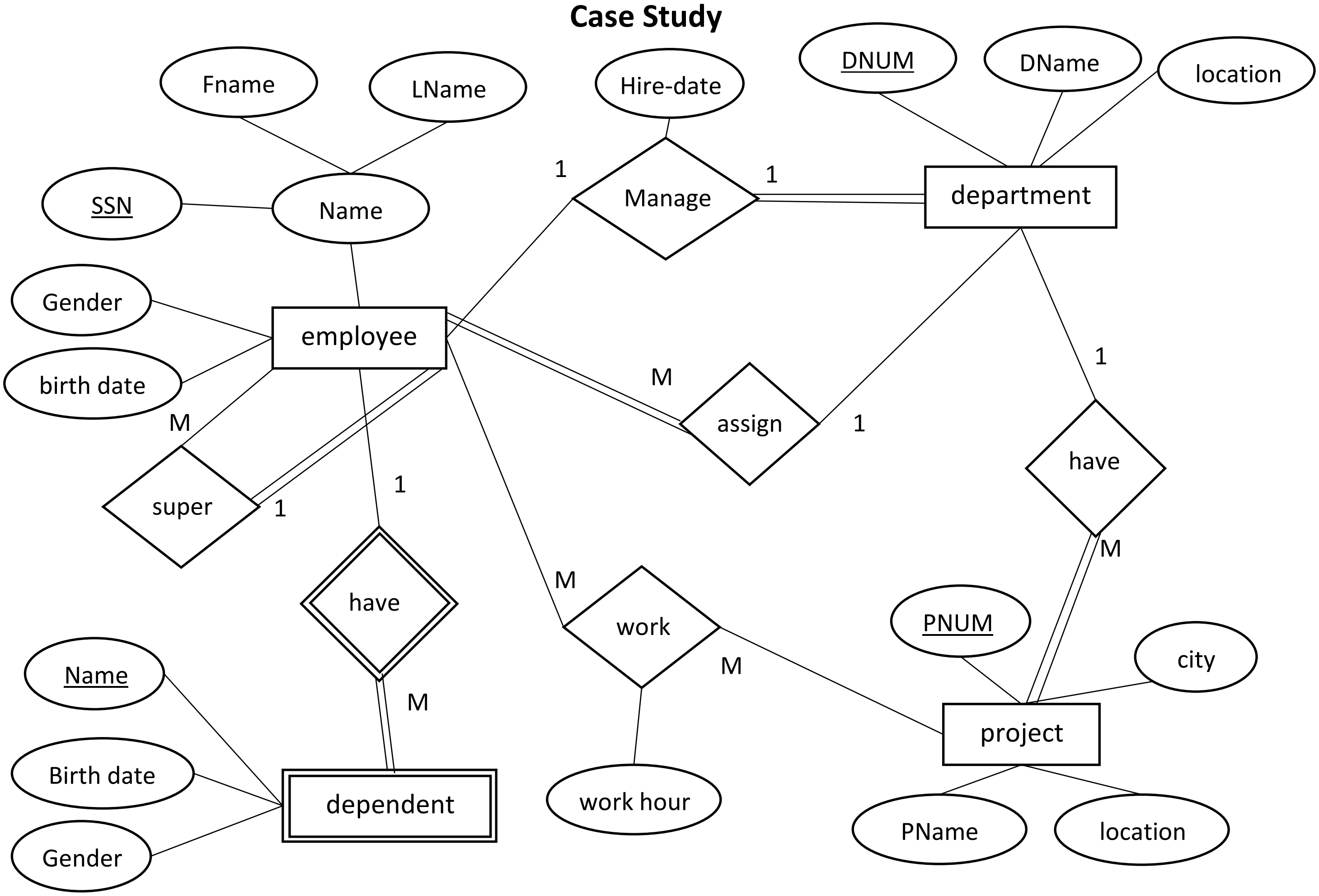


EMPLOYEE

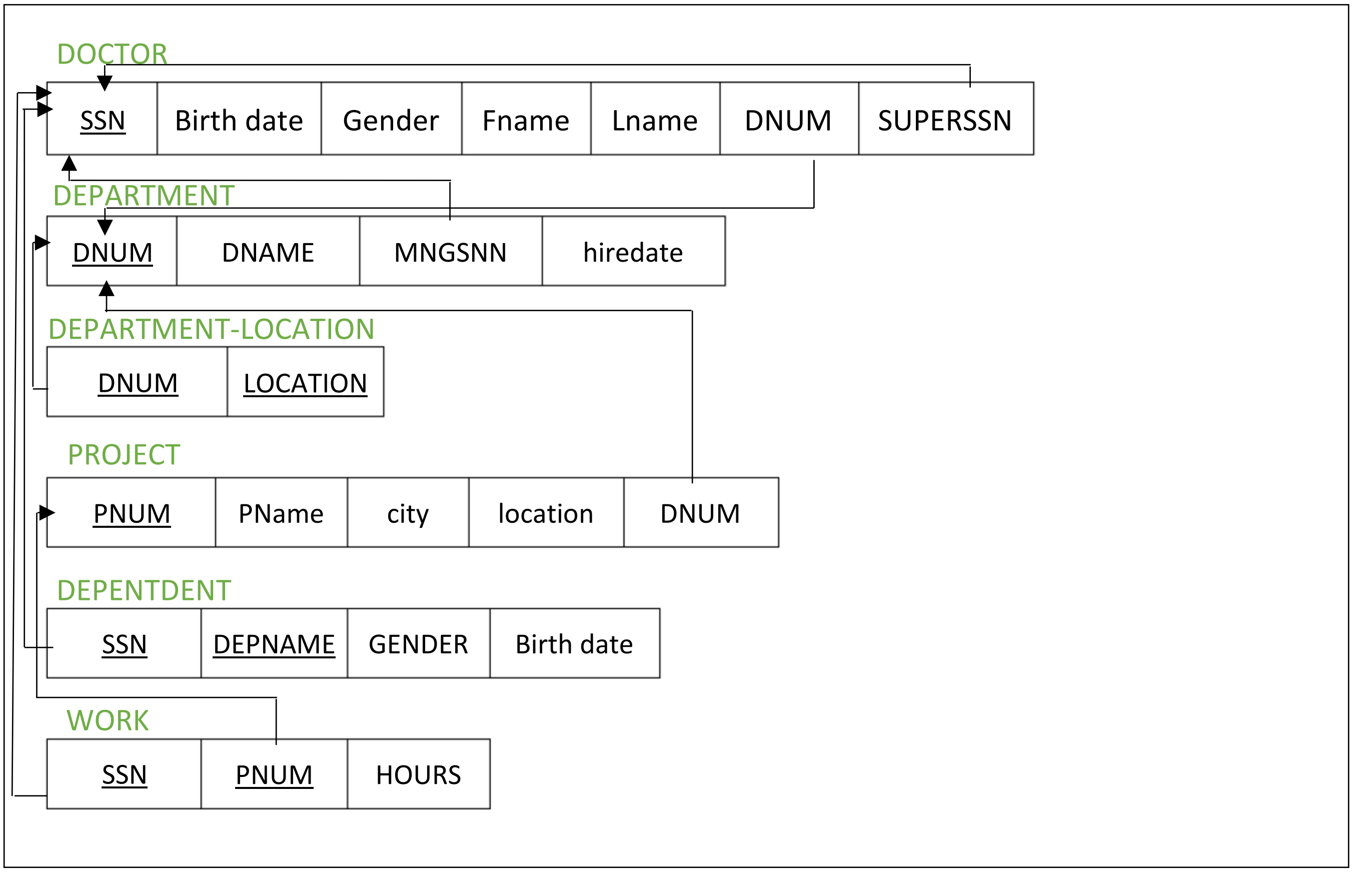
EMP-ID	EMP-NAME	Age	MGR-ID

Primary key

Case Study



Solve



DOCTOR

SSN	Birth date	Gender	Fname	Lname	DNUM	SUPERSSN

DEPARTMENT

DNUM	DNAME	MNGSNN	hiredate

DEPARTMENT-LOCATION

DNUM	LOCATION

PROJECT

PNUM	PName	city	location	DNUM

DEPENTDENT

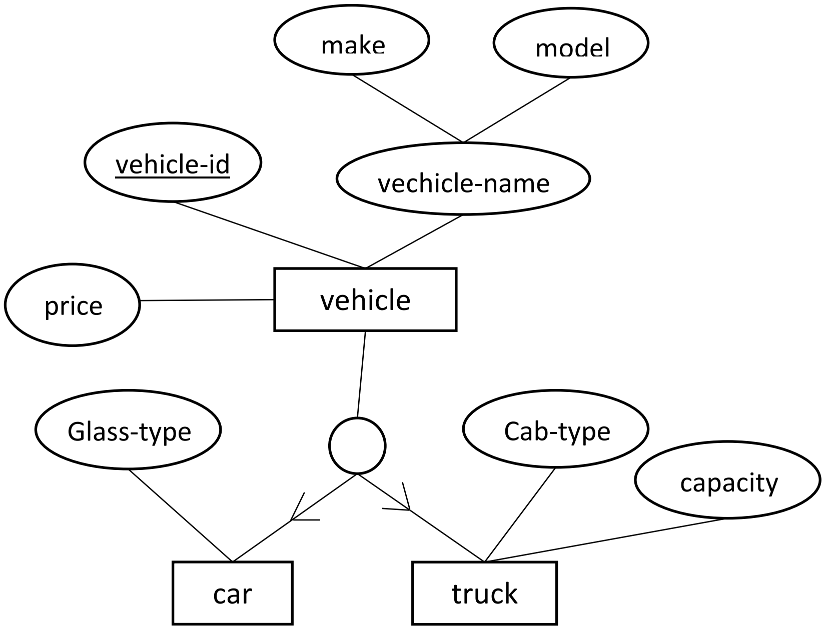
SSN	DEPNAME	GENDER	Birth date

WORK

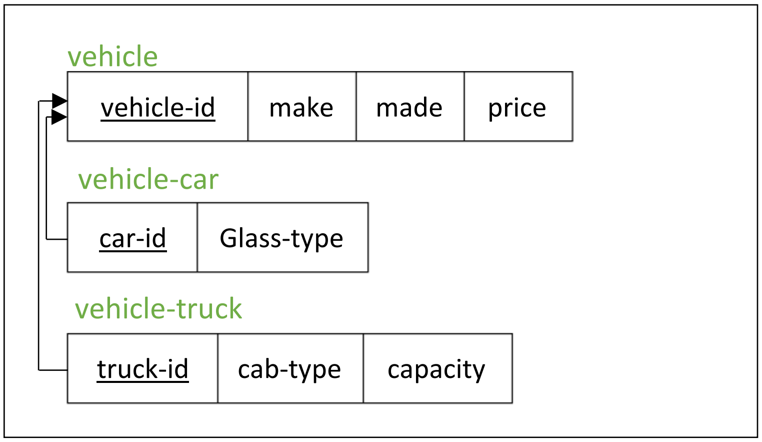
SSN	PNUM	HOURS

EERD Mapping #:

Case Study



Solve



vehicle

<u>vehicle-id</u>	make	made	price

vehicle-car

<u>car-id</u>	Glass-type

vehicle-truck

<u>truck-id</u>	cab-type	capacity

Database Normalization #:

Definition: The process of structuring data to minimize duplication and inconsistencies.

Types of functional dependency:

1- Full Functional Dependency: Attribute is fully Functional Dependency on a PK if its value is determined by the whole PK

2- Partial Functional Dependency: Attribute if has a Partially Functional Dependency on a PK if its value is determined by part of the PK (Composite Key)

3- Transitive Functional Dependency: Attribute is Transitively Functional Dependency on a table if its value is determined by anther non-key attribute which itself determined by PK

Steps of Normalization:

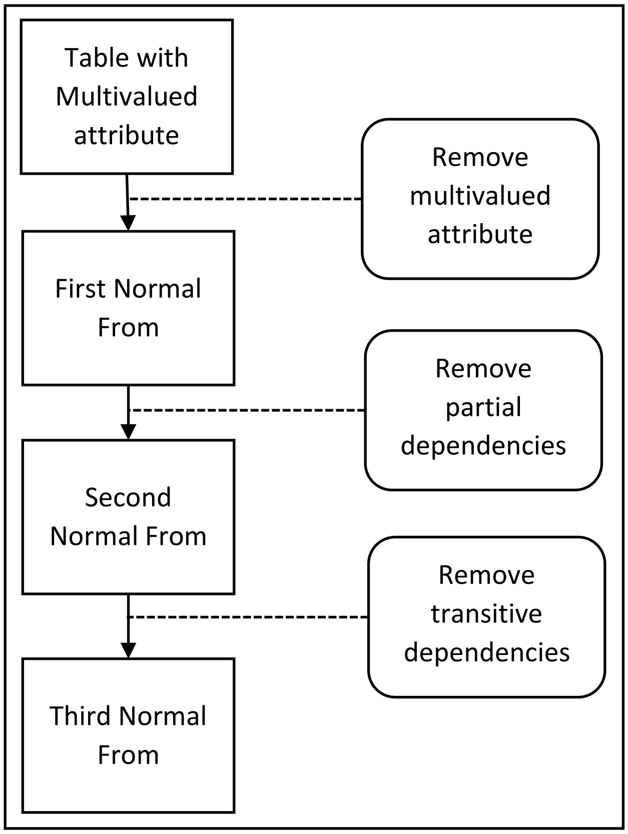


Table with Multivalued attribute
Remove multivalued attribute
First Normal From
Remove partial dependencies
Second Normal From
Remove transitive dependencies
Third Normal From

Example

SID	Sname	Bdate	City	ZipCode	Subject	Grade	Teacher
1	Ahmed	1/1/1980	Cairo	1010	DB	A	Hany
1	Ahmed	1/1/1980	Cairo	1010	DB	B	Eman
1	Ahmed	1/1/1983	Cairo	1010	DB	A	Khalid
2	Ali	1/1/1980	Alex	1111	DB	B	Hany
2	Ali	1/1/1980	Alex	1111	SWE	B	Heba
3	Mohamed	1/1/1990	Cairo	1010	NC	C	Mona

Solve

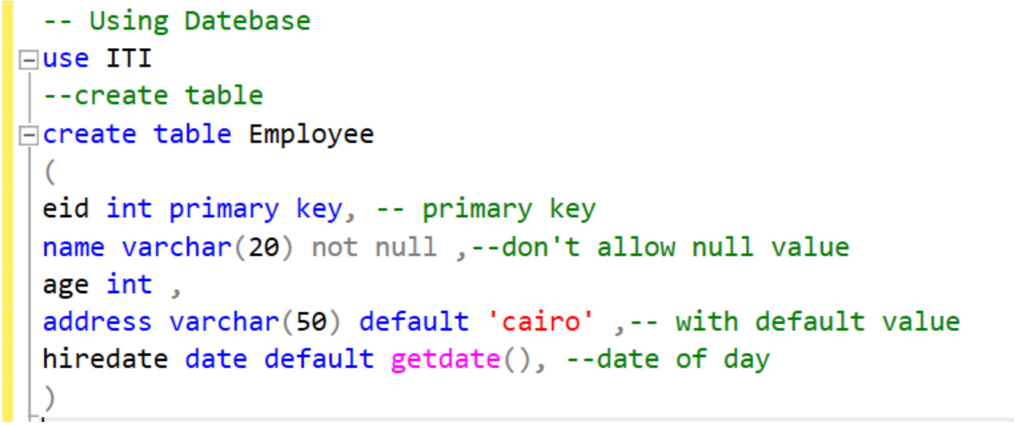
SID	Sname	Bdate	ZipCode
1	Ahmed	1/1/1980	1010
2	Ali	1/1/1983	1010
3	Mohamed	1/1/1980	1111

City	ZipCode
cario	1010
Alex	1111

SID	Subject	Grade
1	DB	A
1	Math	B
1	WinXP	A
2	DB	B
2	SWE	B
3	NC	C

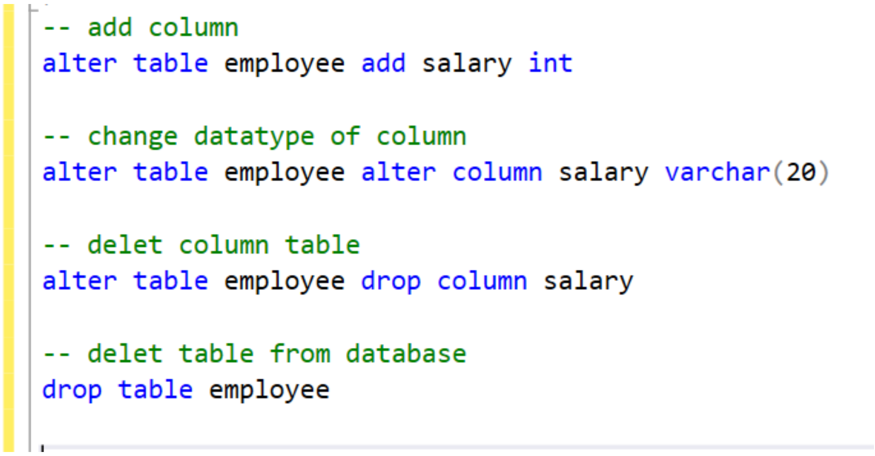
Subject	Teacher
DB1	Hany
Math	Eman
WinXP	Khalid
DB	Hany
SWE	Heba
NC	Mona

1- Use database + create table



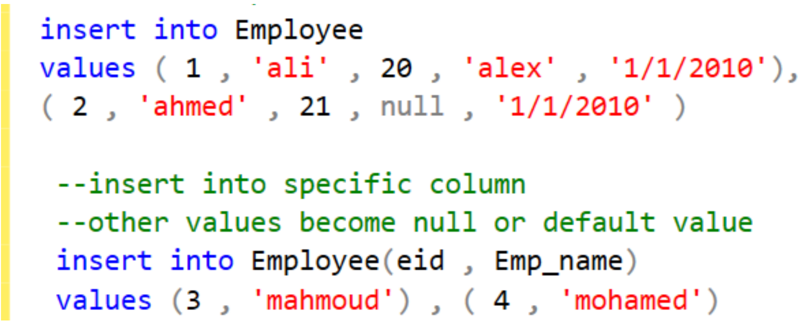
```
-- Using Datebase
use ITI
--create table
create table Employee
(
eid int primary key, -- primary key
name varchar(20) not null ,--don't allow null value
age int ,
address varchar(50) default 'cairo' ,-- with default value
hiredate date default getdate(), --date of day
)
```

2- Edit table: before change any column should Values inside the column must be considered



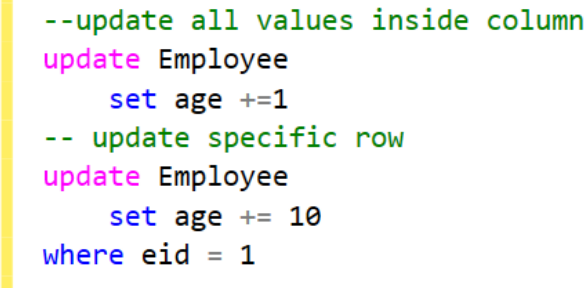
```
-- add column
alter table employee add salary int

-- change datatype of column
alter table employee alter column salary varchar(20)

-- delet column table
alter table employee drop column salary

-- delet table from database
drop table employee
```

3- insert into table: values should match table definition.



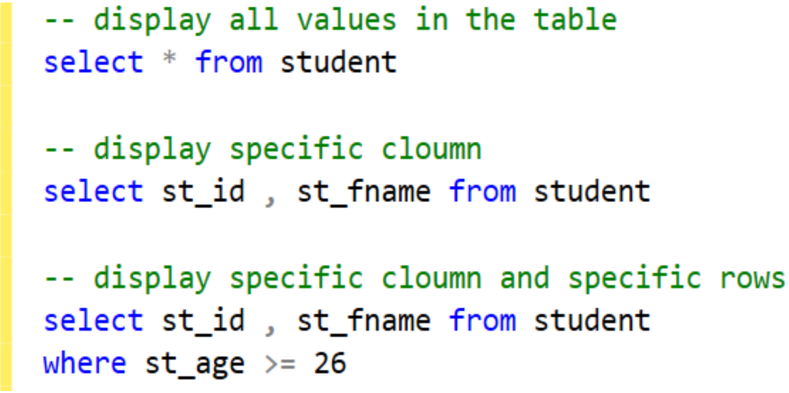
```
insert into Employee
values ( 1 , 'ali' , 20 , 'alex' , '1/1/2010'),
( 2 , 'ahmed' , 21 , null , '1/1/2010' )

 --insert into specific column
 --other values become null or default value
 insert into Employee(eid , Emp_name)
 values (3 , 'mahmoud') , ( 4 , 'mohamed')
```

4- Update values into table:

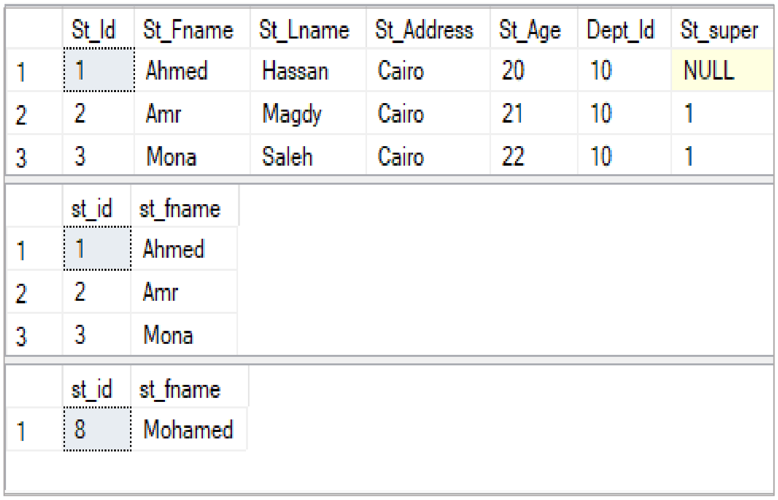


```
--update all values inside column
update Employee
    set age +=1
-- update specific row
update Employee
    set age += 10
where eid = 1
```

5- Display table:



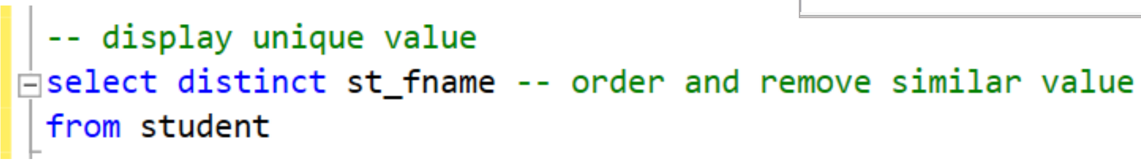
```
-- display all values in the table
select * from student

-- display specific cloumn
select st_id , st_fname from student

-- display specific cloumn and specific rows
select st_id , st_fname from student
where st_age >= 26
```



	St_Id	St_Fname	St_Lname	St_Address	St_Age	Dept_Id	St_super
1	1	Ahmed	Hassan	Cairo	20	10	NULL
2	2	Amr	Magdy	Cairo	21	10	1
3	3	Mona	Saleh	Cairo	22	10	1

	st_id	st_fname
1	1	Ahmed
2	2	Amr
3	3	Mona

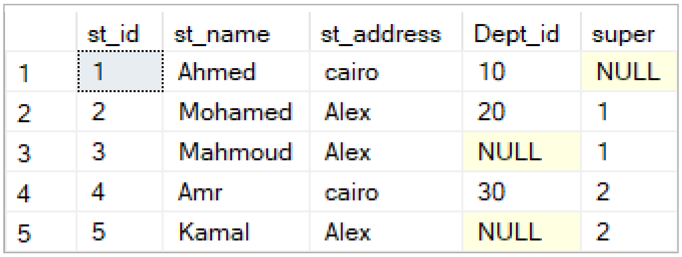
	st_id	st_fname
1	8	Mohamed



```
-- display unique value
select distinct st_fname -- order and remove similar value
from student
```

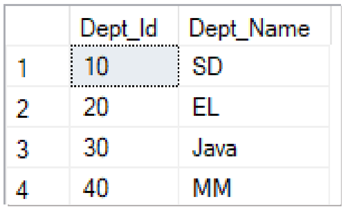
6- Joins: it used to combine data from tables based on common field

Student



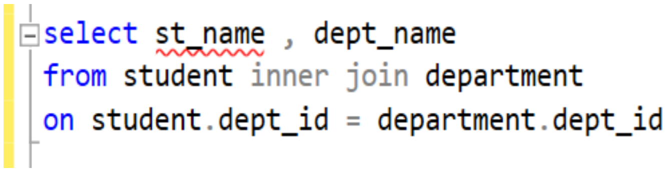
	st_id	st_name	st_address	Dept_id	super
1	1	Ahmed	cairo	10	NULL
2	2	Mohamed	Alex	20	1
3	3	Mahmoud	Alex	NULL	1
4	4	Amr	cairo	30	2
5	5	Kamal	Alex	NULL	2

Department

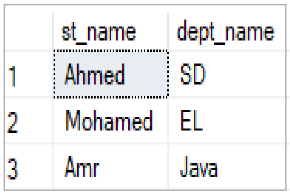


	Dept_Id	Dept_Name
1	10	SD
2	20	EL
3	30	Java
4	40	MM

Inner joins: No Null value on the specific column

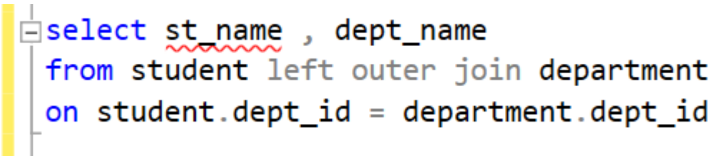


```
select st_name , dept_name
from student inner join department
on student.dept_id = department.dept_id
```

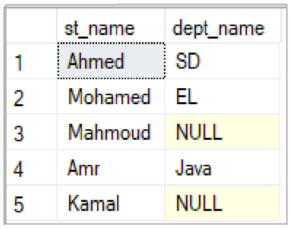


	st_name	dept_name
1	Ahmed	SD
2	Mohamed	EL
3	Amr	Java

Left outer joins:

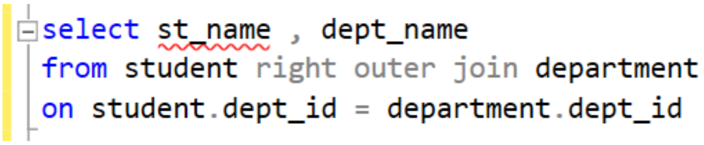


```
select st_name , dept_name
from student left outer join department
on student.dept_id = department.dept_id
```

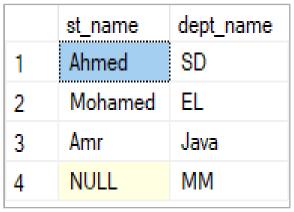


	st_name	dept_name
1	Ahmed	SD
2	Mohamed	EL
3	Mahmoud	NULL
4	Amr	Java
5	Kamal	NULL

Right outer joins:

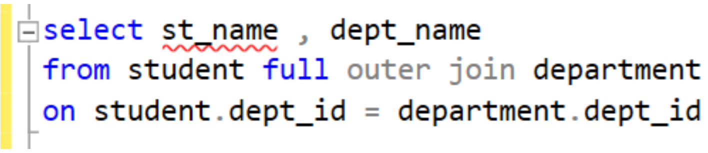


```
select st_name , dept_name
from student right outer join department
on student.dept_id = department.dept_id
```

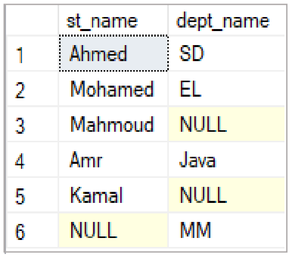


	st_name	dept_name
1	Ahmed	SD
2	Mohamed	EL
3	Amr	Java
4	NULL	MM

Full outer joins:

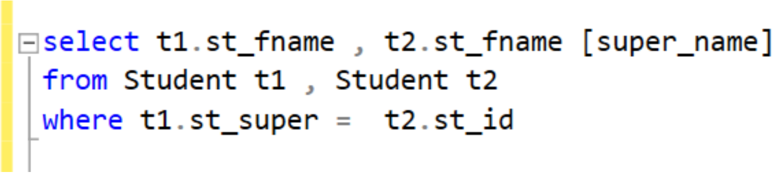


```
select st_name , dept_name
from student full outer join department
on student.dept_id = department.dept_id
```

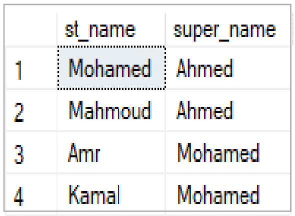


	st_name	dept_name
1	Ahmed	SD
2	Mohamed	EL
3	Mahmoud	NULL
4	Amr	Java
5	Kamal	NULL
6	NULL	MM

Self joins:

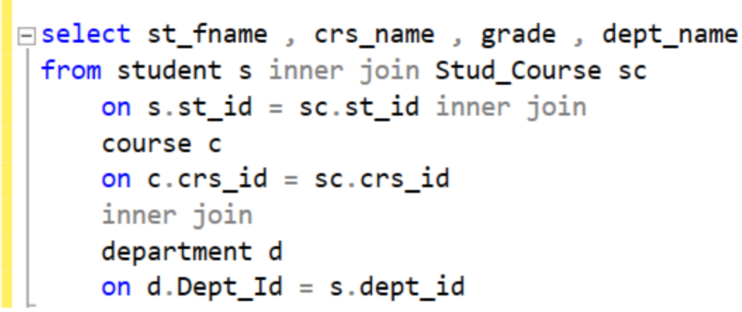


```
select t1.st_fname , t2.st_fname [super_name]
from Student t1 , Student t2
where t1.st_super =  t2.st_id
```



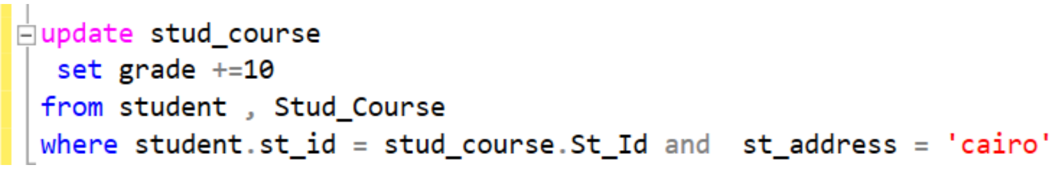
	st_name	super_name
1	Mohamed	Ahmed
2	Mahmoud	Ahmed
3	Amr	Mohamed
4	Kamal	Mohamed

Joins Multiple Table:



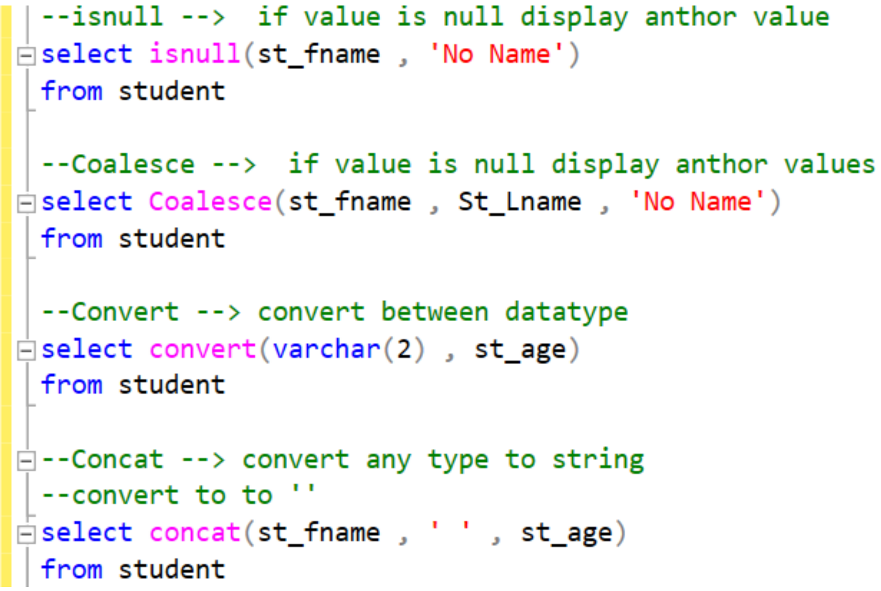
```
select st_fname , crs_name , grade , dept_name
from student s inner join Stud_Course sc
    on s.st_id = sc.st_id inner join
    course c
    on c.crs_id = sc.crs_id
    inner join
    department d
    on d.Dept_Id = s.dept_id
```

Joins + DML:



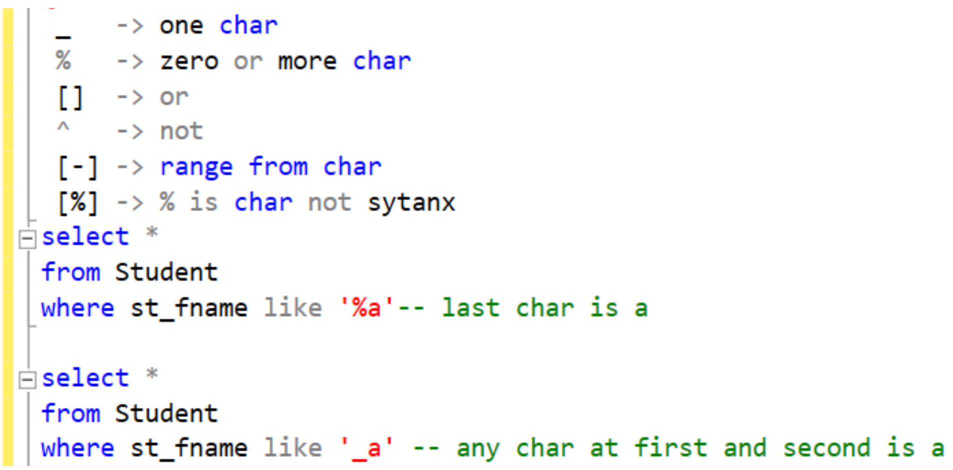
```
update stud_course
 set grade +=10
from student , Stud_Course
where student.st_id = stud_course.St_Id and  st_address = 'cairo'
```

7- Built-in Function:



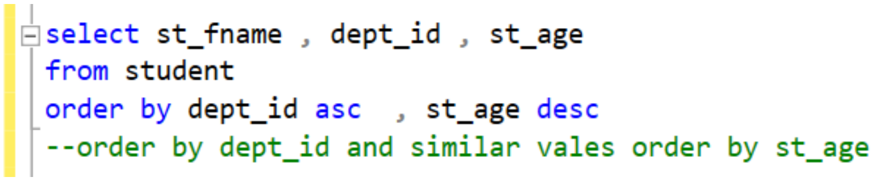
```
--isnull -->  if value is null display anthor value
select isnull(st_fname , 'No Name')
from student

--Coalesce -->  if value is null display anthor values
select Coalesce(st_fname , St_Lname , 'No Name')
from student

--Convert --> convert between datatype
select convert(varchar(2) , st_age)
from student

--Concat --> convert any type to string
--convert to to ''
select concat(st_fname , ' ' , st_age)
from student
```

Like:



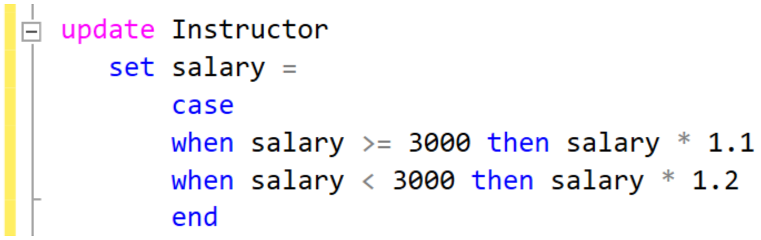
```
_   -> one char
%   -> zero or more char
[]  -> or
^   -> not
[-] -> range from char
[%] -> % is char not sytanx
select *
from Student
where st_fname like '%a'-- last char is a

select *
from Student
where st_fname like '_a' -- any char at first and second is a
```

Order by:



```
select st_fname , dept_id , st_age
from student
order by dept_id asc  , st_age desc
--order by dept_id and similar vales order by st_age
```

Case:



```
update Instructor
    set salary =
        case
        when salary >= 3000 then salary * 1.1
        when salary < 3000 then salary * 1.2
        end
```

SQL Constraints #:

Definition: rules or restrictions enforced on the data in a database table to ensure data integrity and consistency. These constraints define what type of data is valid for specific columns or the relationships between different tables, Specific object per table.

Types:

1- NOT NULL: Ensures that a column cannot have a NULL value.

2- UNIQUE: Ensures that all the values in a column are unique.

3- PRIMARY KEY: A combination of NOT NULL and UNIQUE. Ensures that a column (or a group of columns) uniquely identifies each row in the table.

4- FOREIGN KEY: Establishes a link between the data in two tables. It enforces referential integrity by ensuring that a value in one table corresponds to a value in another table.

5- CHECK: Ensures that all values in a column satisfy a specific condition.

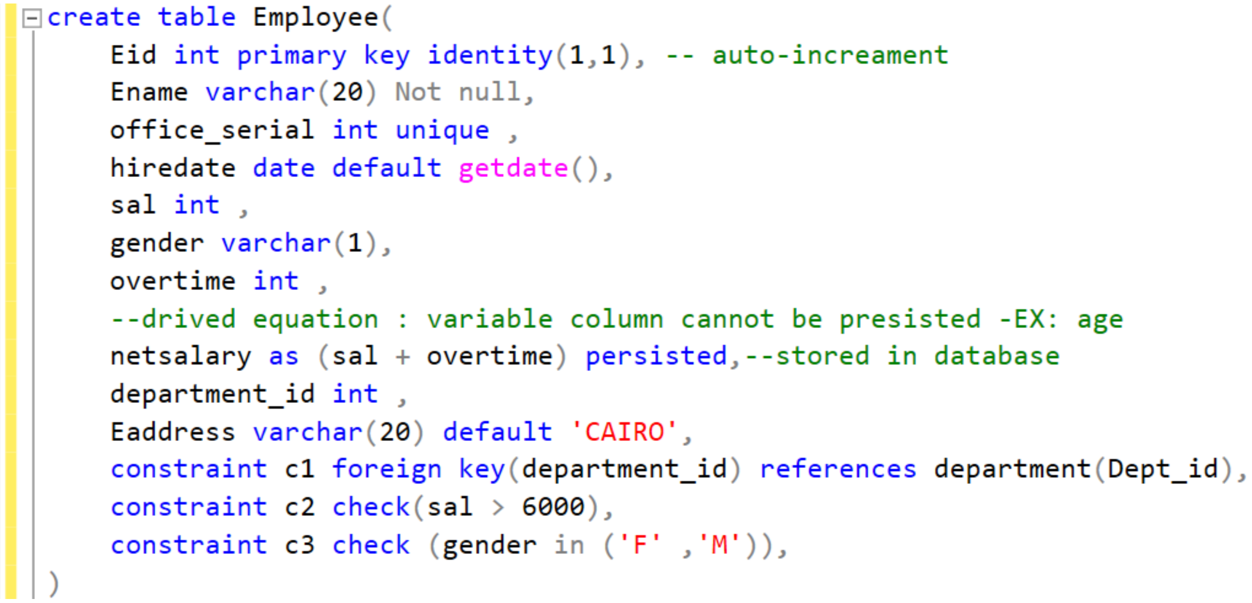
6- DEFAULT: Sets a default value for a column when no value is specified.

7- INDEX: Improves the performance of queries by creating indexes on columns (not technically a constraint but related to optimization).

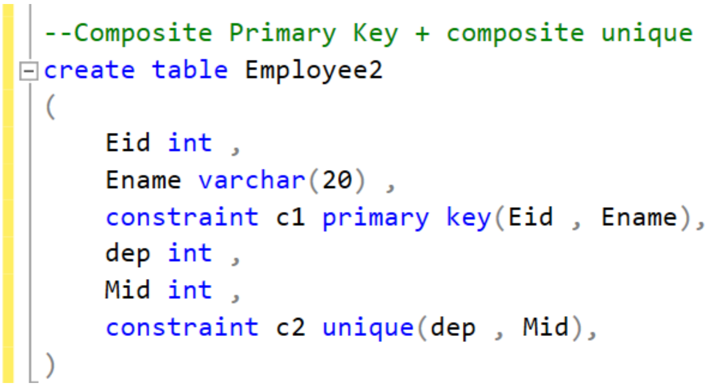
8- AUTO_INCREMENT: Used to automatically generate a unique number for a column (mostly for primary keys).

9- Composite Key: A primary key consisting of multiple columns.

10- ON DELETE/ON UPDATE (Referential Actions): Specifies actions when a referenced row is deleted or updated.



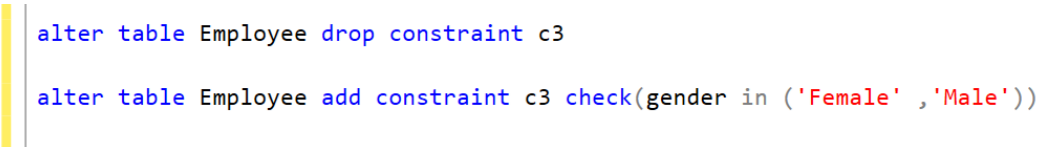
```sql
create table Employee(
    Eid int primary key identity(1,1), -- auto-increament
    Ename varchar(20) Not null,
    office_serial int unique ,
    hiredate date default getdate(),
    sal int ,
    gender varchar(1),
    overtime int ,
    --drived equation : variable column cannot be presisted -EX: age
    netsalary as (sal + overtime) persisted,--stored in database
    department_id int ,
    Eaddress varchar(20) default 'CAIRO',
    constraint c1 foreign key(department_id) references department(Dept_id),
    constraint c2 check(sal > 6000),
    constraint c3 check (gender in ('F' ,'M')),
)
```



```sql
--Composite Primary Key + composite unique
create table Employee2
(
    Eid int ,
    Ename varchar(20) ,
    constraint c1 primary key(Eid , Ename),
    dep int ,
    Mid int ,
    constraint c2 unique(dep , Mid),
)
```

- Add Constraints or drop Constraints:

Check if your old data complies with the new constraint. If it doesn't, you'll need to clean up or update the data.



```sql
alter table Employee drop constraint c3

alter table Employee add constraint c3 check(gender in ('Female' ,'Male'))
```

Options for Foreign key constraints:

1- ON DELETE:

CASCADE: Deletes child rows when the parent row is deleted.

SET NULL: Sets the foreign key column in child rows to NULL.

SET DEFAULT: Sets the foreign key column in child rows to its default value.

NO ACTION (default): Prevents deletion of the parent row if child rows exist.

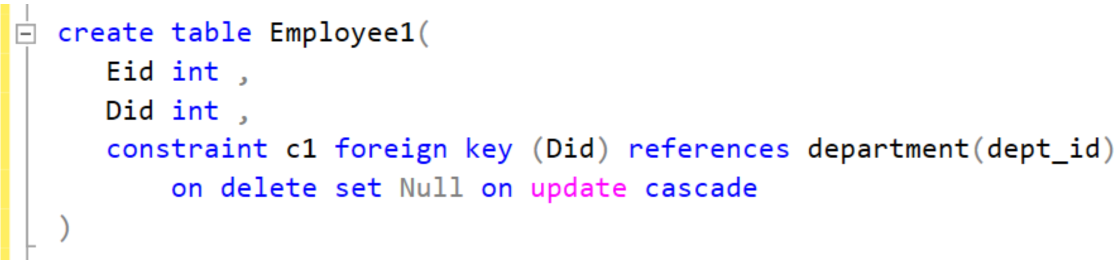
2- ON UPDATE

CASCADE: Updates the foreign key column in child rows when the parent key is updated.

SET NULL: Sets the foreign key column in child rows to NULL.

SET DEFAULT: Sets the foreign key column in child rows to its default value.

NO ACTION (default): Prevents updating the parent key if child rows exist.



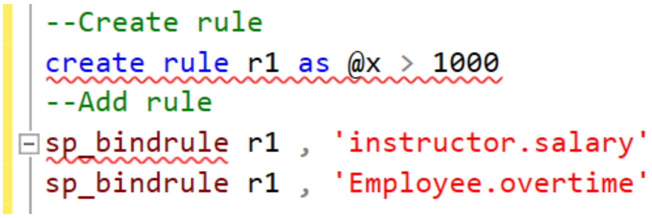
```sql
create table Employee1(
    Eid int ,
    Did int ,
    constraint c1 foreign key (Did) references department(dept_id)
        on delete set Null on update cascade
)
```

Rule #:

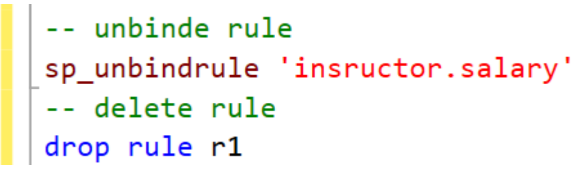
Definition: Create a new data type with constraints -> database level (schema level)

path: databases \ database_name \ programmability \ rules

- rule not applied to old data

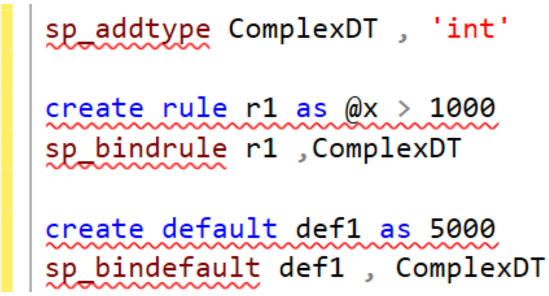


```
--Create rule
create rule r1 as @x > 1000
--Add rule
sp_bindrule r1 , 'instructor.salary'
sp_bindrule r1 , 'Employee.overtime'
```



```
-- unbinde rule
sp_unbindrule 'insructor.salary'
-- delete rule
drop rule r1
```

--create a new datatype with constraints



```
sp_addtype ComplexDT , 'int'

create rule r1 as @x > 1000
sp_bindrule r1 ,ComplexDT

create default def1 as 5000
sp_bindefault def1 , ComplexDT
```

Aggregte Function #:

Definition: Parfum A chain of two that A six F money when returning a single messenger

Common aggregate function: 1- count 2- sum 3- avg 4- min 5-max

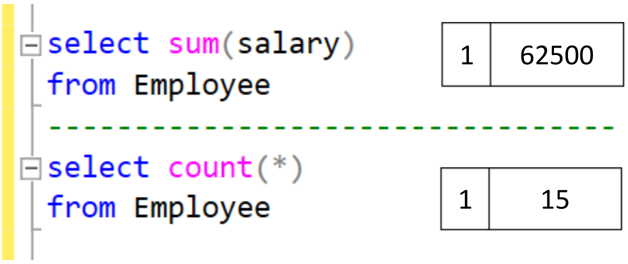
-Aggregate function didn't take null values into account.

Employee

Eid	Ename	Salary	EAddress	did
1	ahmed	3000	cairo	10
2	ali	5000	cairo	10
3	eman	2000	cairo	10
4	khalid	1000	alex	10
5	yousef	4000	alex	10
6	sameh	5000	alex	10
7	mohamed	6000	alex	20
8	alaa	7000	alex	20
9	ola	4000	cairo	20
10	reem	2000	cairo	20
11	nada	9000	cairo	20
12	sayed	8000	mansoura	30
13	reham	1500	mansoura	30
14	sally	2000	mansoura	30
15	omar	3000	mansoura	30

Departments

Did	Dname
1	DP1
2	DP2
3	DP3



```
select sum(salary)
from Employee
--------------------------------
select count(*)
from Employee
```

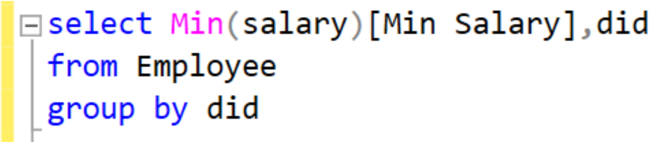
1	62500

1	15

Group by #:

Definition: clause is used to group rows that have the same values into summary rows.

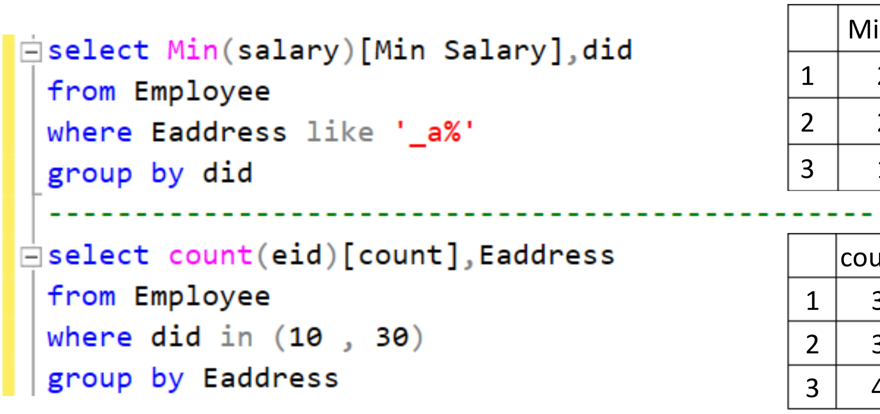
group:



```
select Min(salary)[Min Salary],did
from Employee
group by did
```

	Min Salary	did
1	1000	10
2	2000	20
3	3000	30

group + where: where filter to rows

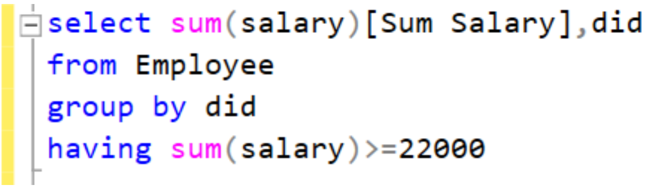


```
select Min(salary)[Min Salary],did
from Employee
where Eaddress like '_a%'
group by did
---------------------------------------------------
select count(eid)[count],Eaddress
from Employee
where did in (10 , 30)
group by Eaddress
```

	Min Salary	did
1	2000	10
2	2000	20
3	1500	30

	count	Eaddress
1	3	alex
2	3	cario
3	4	mansoura

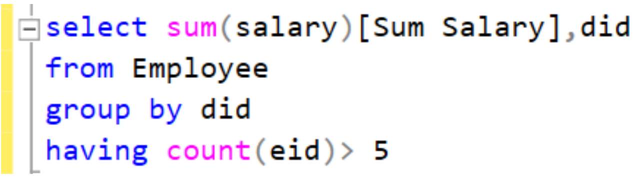
having: clause is used to filter grouped data.



```
select sum(salary)[Sum Salary],did
from Employee
group by did
having sum(salary)>=22000
```

	Sum Salary	did
1	20000	10

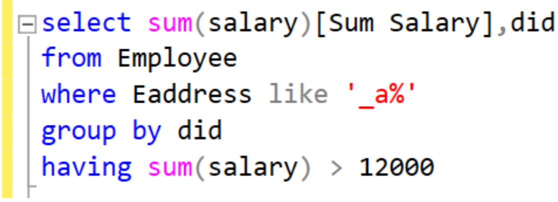
Having may be different with select



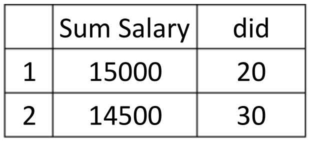
```
select sum(salary)[Sum Salary],did
from Employee
group by did
having count(eid)> 5
```

	Sum Salary	did
1	20000	10

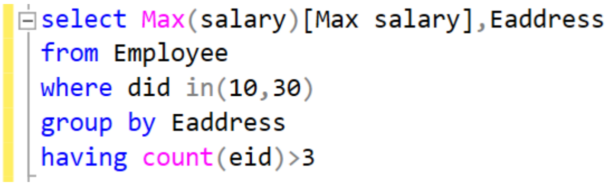
having + where:



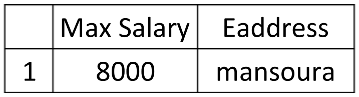
```
select sum(salary)[Sum Salary],did
from Employee
where Eaddress like '_a%'
group by did
having sum(salary) > 12000
```



	Sum Salary	did
1	15000	20
2	14500	30

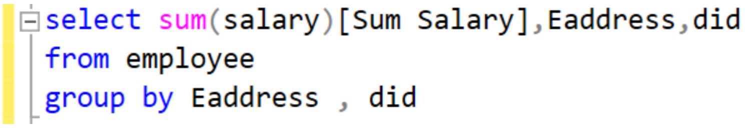


```
select Max(salary)[Max salary],Eaddress
from Employee
where did in(10,30)
group by Eaddress
having count(eid)>3
```

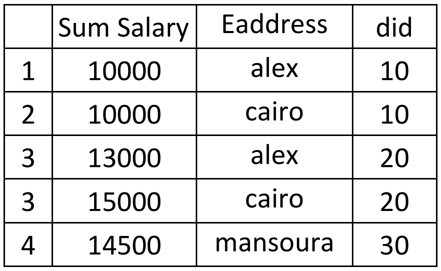


	Max Salary	Eaddress
1	8000	mansoura

Group by two columns:

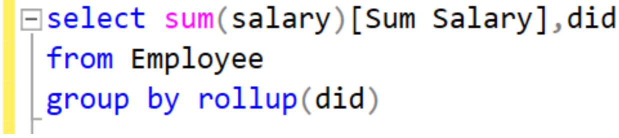


```
select sum(salary)[Sum Salary],Eaddress,did
from employee
group by Eaddress , did
```

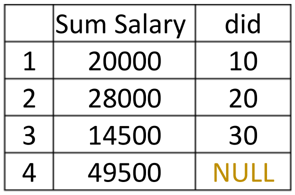


	Sum Salary	Eaddress	did
1	10000	alex	10
2	10000	cairo	10
3	13000	alex	20
3	15000	cairo	20
4	14500	mansoura	30

group by rollup: including subtotals and a grand total

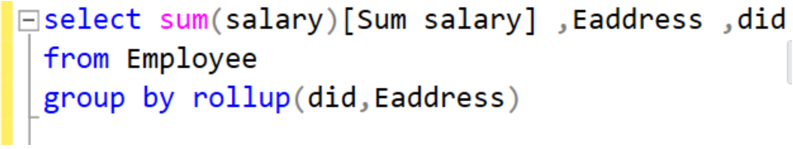


```
select sum(salary)[Sum Salary],did
from Employee
group by rollup(did)
```

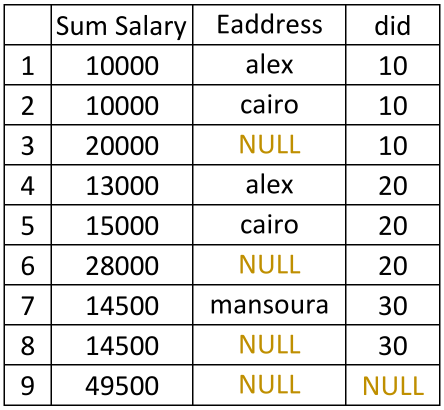


	Sum Salary	did
1	20000	10
2	28000	20
3	14500	30
4	49500	NULL

group by rollup (two column):

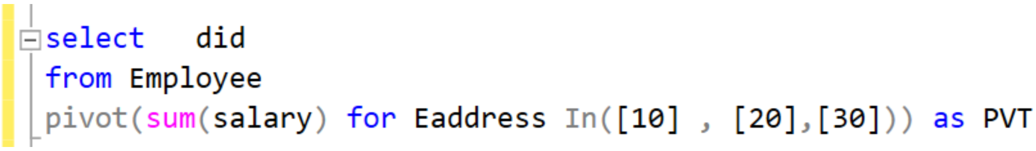


```
select sum(salary)[Sum salary] ,Eaddress ,did
from Employee
group by rollup(did,Eaddress)
```



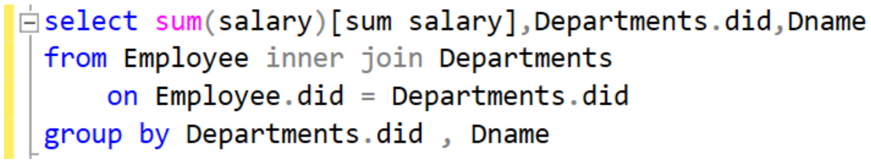
	Sum Salary	Eaddress	did
1	10000	alex	10
2	10000	cairo	10
3	20000	NULL	10
4	13000	alex	20
5	15000	cairo	20
6	28000	NULL	20
7	14500	mansoura	30
8	14500	NULL	30
9	49500	NULL	NULL

pivot: =Group by two column as a matrix

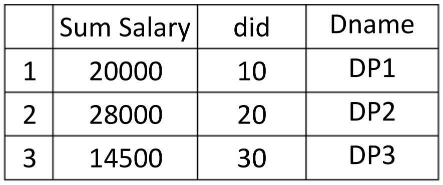


```
select   did
from Employee
pivot(sum(salary) for Eaddress In([10] , [20],[30])) as PVT
```

Join + Group by:



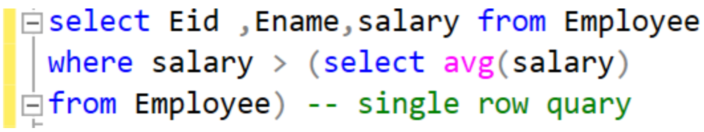
```sql
select sum(salary)[sum salary],Departments.did,Dname
from Employee inner join Departments
     on Employee.did = Departments.did
group by Departments.did , Dname
```



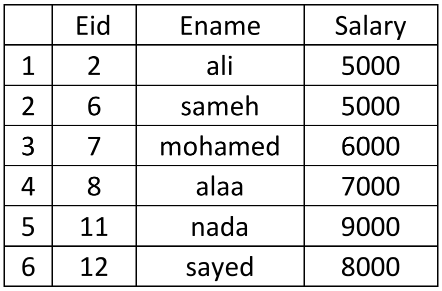
	Sum Salary	did	Dname
1	20000	10	DP1
2	28000	20	DP2
3	14500	30	DP3

Subqueris #:

Definition: are nested queries that are placed within another SQL query. They are enclosed in parentheses and can be used in various parts of a query.

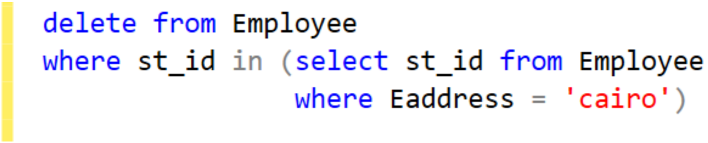


```sql
select Eid ,Ename,salary from Employee
where salary > (select avg(salary)
from Employee) -- single row quary
```



	Eid	Ename	Salary
1	2	ali	5000
2	6	sameh	5000
3	7	mohamed	6000
4	8	alaa	7000
5	11	nada	9000
6	12	sayed	8000

Subqueries + DML:

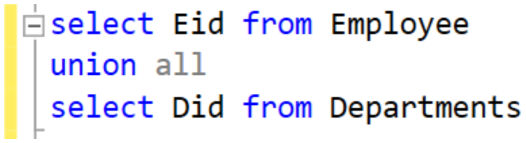


```sql
delete from Employee
where st_id in (select st_id from Employee
                where Eaddress = 'cairo')
```

Union Family #:

Definition: operator in SQL is used to combine the result sets of two or more SELECT queries.

Union all: combine the results of two or more SELECT queries without removing duplicate rows.



```
select Eid from Employee
union all
select Did from Departments
```

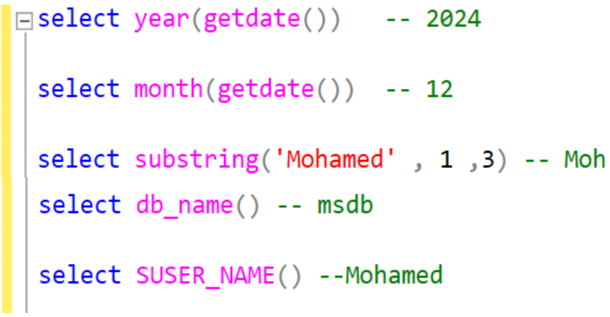
	Eid
1	1
2	2
3	3
	…
16	10
17	20
18	30

Union: combine the results of two or more SELECT queries with Order row and removing duplicate rows.

intersect: used to return only the rows that are common between the result sets of two SELECT queries.

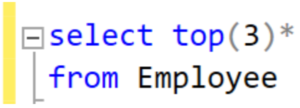
Except: used to return rows from the first SELECT query that are not present in the second SELECT query.

Buit-in funtion #:



```
select year(getdate())   -- 2024

select month(getdate())  -- 12

select substring('Mohamed' , 1 ,3) -- Moh
select db_name() -- msdb

select SUSER_NAME() --Mohamed
```

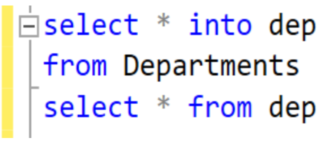
Top:



```
select top(3)*
from Employee
```

	Eid	Ename	Salary	address	did
1	1	ahmed	3000	cairo	10
2	2	ali	5000	cairo	10
3	3	eman	2000	cairo	10

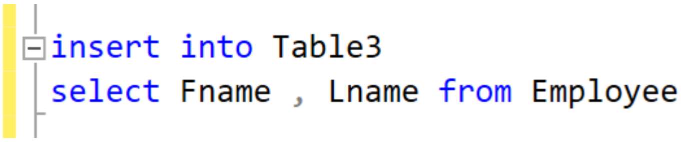
Into: statement is used to create a new table and populate it with data from an existing table or query.



```
select * into dep
from Departments
select * from dep
```

Did	Dname
1	DP1
2	DP2
3	DP3

-Copy values from table and past on another table.



```
insert into Table3
select Fname , Lname from Employee
```

Ranking Function #:

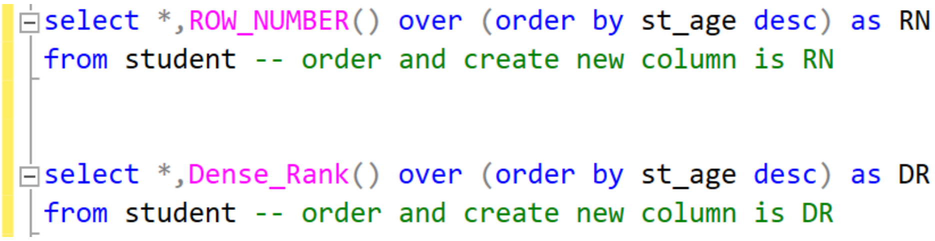
Definition: operator in SQL is used to combine the result sets of two or more SELECT queries

ROW_NUMBER(): Assigns a unique sequential number to each row within a partition.

DENSE_RANK(): Assigns a rank to rows, with ties receiving the same rank. The next rank does not skip.

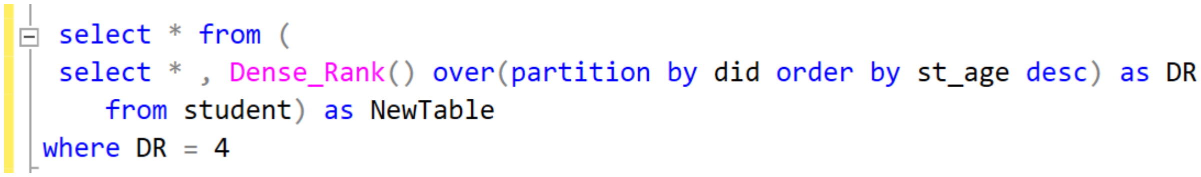
NTILE(n): Divides rows into n groups and assigns a group number to each row.

RANK(): Assigns a rank to rows, with ties receiving the same rank. The next rank skips as per the number of ties.



```
select *,ROW_NUMBER() over (order by st_age desc) as RN
from student -- order and create new column is RN


select *,Dense_Rank() over (order by st_age desc) as DR
from student -- order and create new column is DR
```

Partition by: divide the result set into groups, or partitions, before the window function is applied.

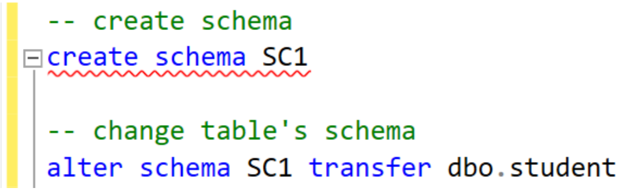


```
 select * from (
 select * , Dense_Rank() over(partition by did order by st_age desc) as DR
    from student) as NewTable
where DR = 4
```

Schema #:

Definition: logical container or namespace that holds database objects such as tables, views, stored procedures, and other objects. It provides a way to organize and manage these objects within a database for better clarity, security, and access control.

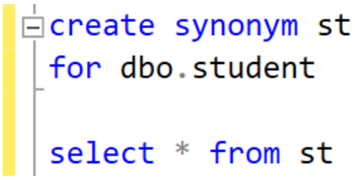
Path: Databases / database_name / security / schemas



```
-- create schema
create schema SC1

-- change table's schema
alter schema SC1 transfer dbo.student
```

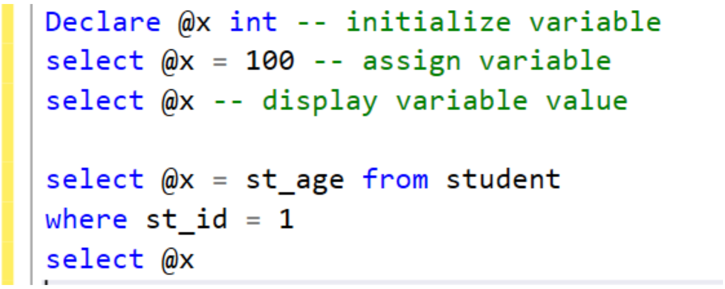
Synonyms #:

Definition: allows you to reference a target object with a different name, which can simplify access or improve maintainability.



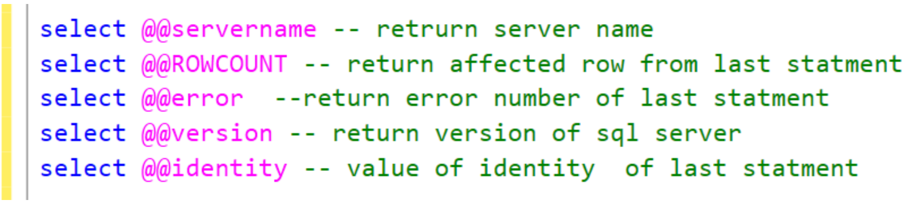
```
create synonym st
for dbo.student

select * from st
```

Variables #:



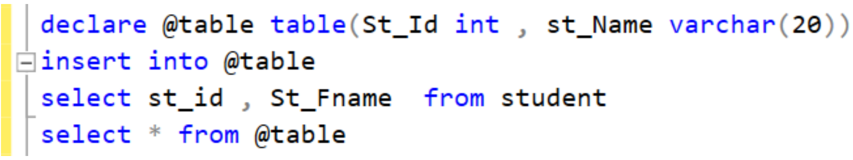
```
Declare @x int -- initialize variable
select @x = 100 -- assign variable
select @x -- display variable value

select @x = st_age from student
where st_id = 1
select @x
```

Global variable: can’t declare global variable – can’t assign global variable



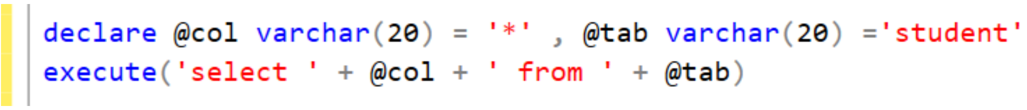
```
select @@servername -- retrurn server name
select @@ROWCOUNT -- return affected row from last statment
select @@error  --return error number of last statment
select @@version -- return version of sql server
select @@identity -- value of identity  of last statment
```

Declare table:



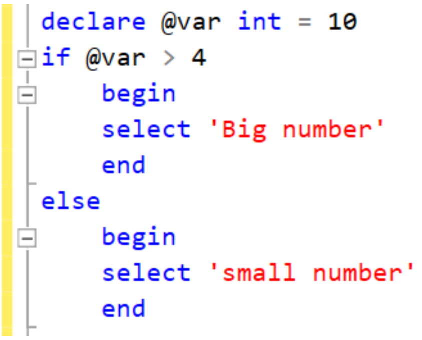
```
declare @table table(St_Id int , st_Name varchar(20))
insert into @table
select st_id , St_Fname  from student
select * from @table
```

Dynamic query:



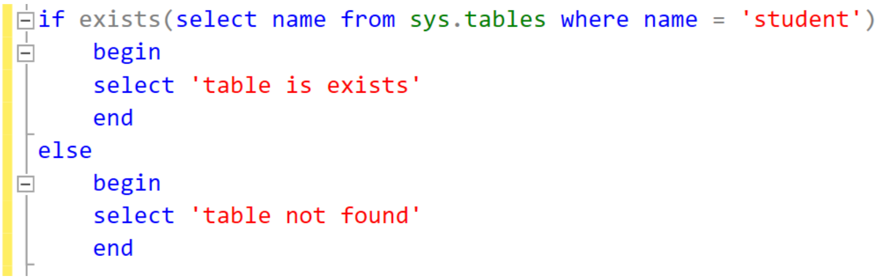
```
declare @col varchar(20) = '*' , @tab varchar(20) ='student'
execute('select ' + @col + ' from ' + @tab)
```

Control of flow statement #:



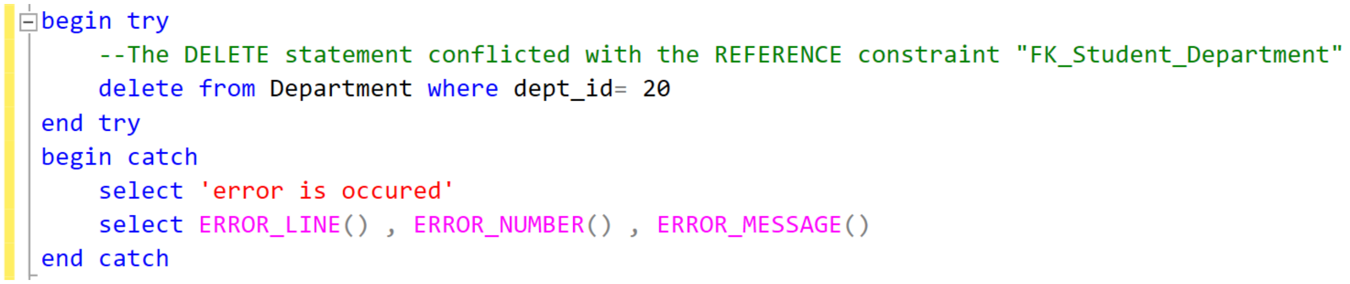
```
declare @var int = 10
if @var > 4
    begin
    select 'Big number'
    end
else
    begin
    select 'small number'
    end
```

exists, not exists: check if the select query return value or not



```
if exists(select name from sys.tables where name = 'student')
    begin
    select 'table is exists'
    end
else
    begin
    select 'table not found'
    end
```

try, catch: handle error

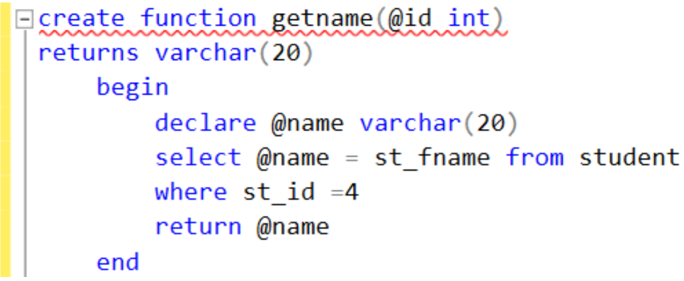


```
begin try
    --The DELETE statement conflicted with the REFERENCE constraint "FK_Student_Department"
    delete from Department where dept_id= 20
end try
begin catch
    select 'error is occured'
    select ERROR_LINE() , ERROR_NUMBER() , ERROR_MESSAGE()
end catch
```

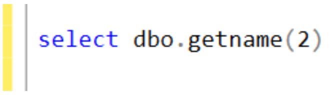
User Defined Function#:

Path: Databases / database_name / programmability / Functions

Scalar Function: These functions return a single value of a specific data type.

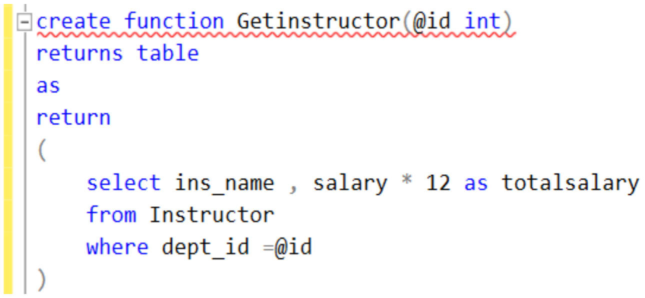


```
create function getname(@id int)
returns varchar(20)
    begin
        declare @name varchar(20)
        select @name = st_fname from student
        where st_id =4
        return @name
    end
```

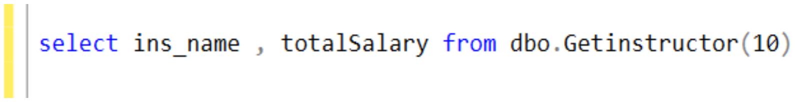


```
select dbo.getname(2)
```

Inline table Function: These return a table and are defined with a single SELECT statement.

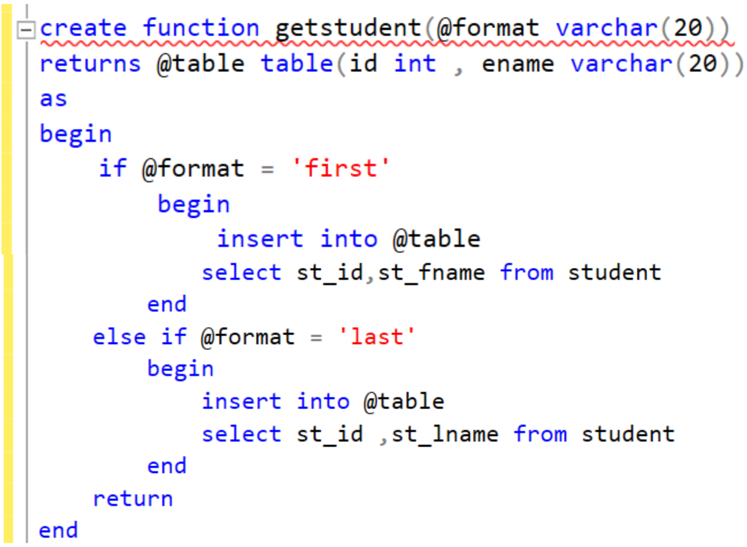


```
create function Getinstructor(@id int)
returns table
as
return
(
    select ins_name , salary * 12 as totalsalary
    from Instructor
    where dept_id =@id
)
```

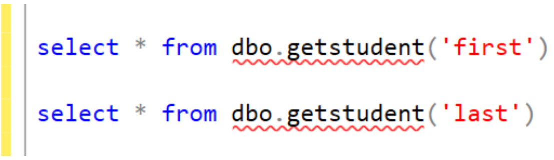


```
select ins_name , totalSalary from dbo.Getinstructor(10)
```

Multi-Statement Table-Valued Functions: These return a table but allow multiple statements in the function body to populate it.



```
create function getstudent(@format varchar(20))
returns @table table(id int , ename varchar(20))
as
begin
    if @format = 'first'
        begin
            insert into @table
            select st_id,st_fname from student
        end
    else if @format = 'last'
        begin
            insert into @table
            select st_id ,st_lname from student
        end
    return
end
```

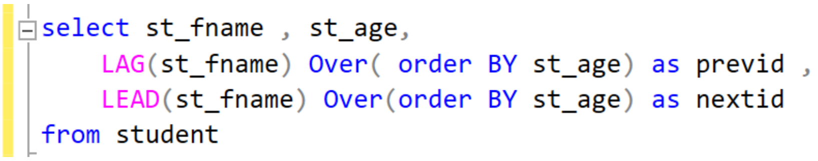


```
select * from dbo.getstudent('first')

select * from dbo.getstudent('last')
```

LAG, LEDD Function#:

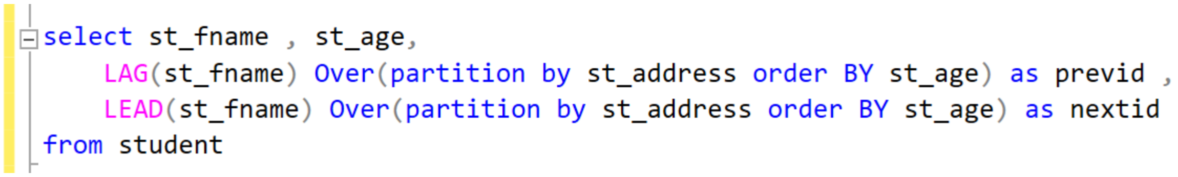
LAG: Retrieves the value of a column from a previous row**.**

LEAD: Retrieves the value of a column from a subsequent row.



```
select st_fname , st_age,
    LAG(st_fname) Over( order BY st_age) as previd ,
    LEAD(st_fname) Over(order BY st_age) as nextid
from student
```

LAG, LEAD + Partition BY: order in same partition first then LAG, LEAD in each row

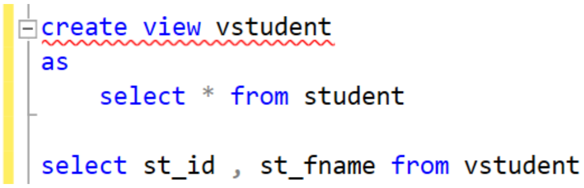


```
select st_fname , st_age,
    LAG(st_fname) Over(partition by st_address order BY st_age) as previd ,
    LEAD(st_fname) Over(partition by st_address order BY st_age) as nextid
from student
```

First + Last: Retrieves the value of a column from a first and Last row**.**

view #:

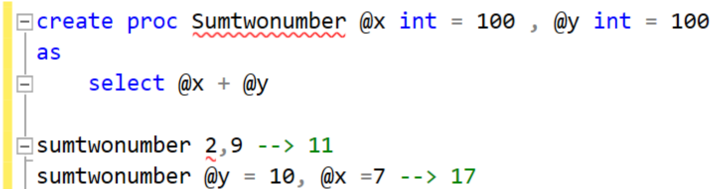
Definition: is a virtual table based on the result of a query. It does not store data itself but retrieves it dynamically from the underlying tables whenever accessed



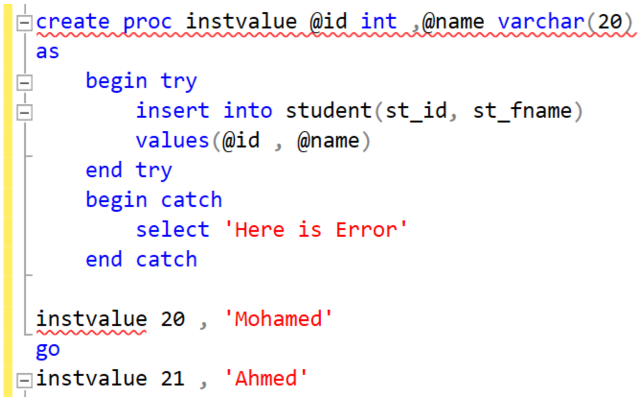
```
create view vstudent
as
    select * from student

select st_id , st_fname from vstudent
```

Stored procedure #:

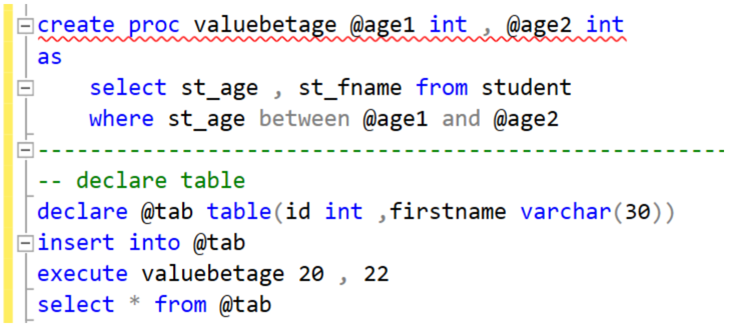
Definition: precompiled collection of SQL statements and optional control-of-flow logic. Stored procedures are used to encapsulate complex queries, enforce business logic, and improve performance by reducing the overhead of query parsing and execution.



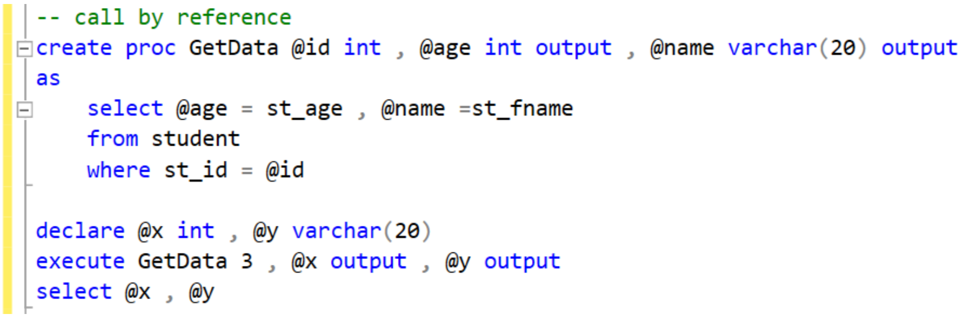
```
create proc Sumtwonumber @x int = 100 , @y int = 100
as
    select @x + @y

sumtwonumber 2,9 --> 11
sumtwonumber @y = 10, @x =7 --> 17
```



```
create proc instvalue @id int ,@name varchar(20)
as
    begin try
        insert into student(st_id, st_fname)
        values(@id , @name)
    end try
    begin catch
        select 'Here is Error'
    end catch

instvalue 20 , 'Mohamed'
go
instvalue 21 , 'Ahmed'
```



```
create proc valuebetage @age1 int , @age2 int
as
    select st_age , st_fname from student
    where st_age between @age1 and @age2
----------------------------------------------------
-- declare table
declare @tab table(id int ,firstname varchar(30))
insert into @tab
execute valuebetage 20 , 22
select * from @tab
```

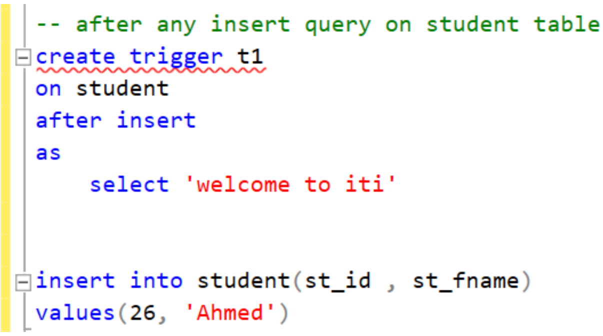


```
-- call by reference
create proc GetData @id int , @age int output , @name varchar(20) output
as
    select @age = st_age , @name =st_fname
    from student
    where st_id = @id

declare @x int , @y varchar(20)
execute GetData 3 , @x output , @y output
select @x , @y
```

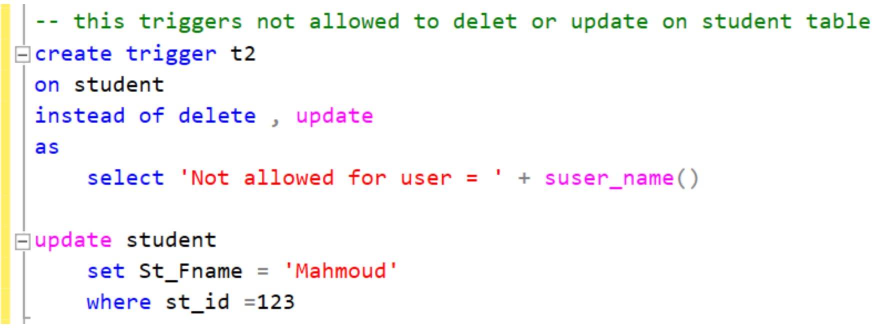
Triggers #:

Definition: special types of stored procedures that automatically execute in response to certain events on a table or a view. Triggers can be used for enforcing business rules, auditing changes, or automatically propagating changes to other tables.

Path: Databases / database_name / table_name / Triggers

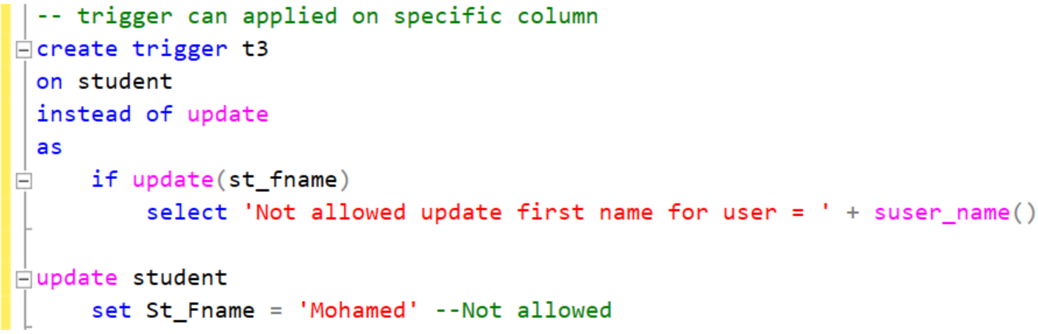


```
-- after any insert query on student table
create trigger t1
on student
after insert
as
    select 'welcome to iti'


insert into student(st_id , st_fname)
values(26, 'Ahmed')
```

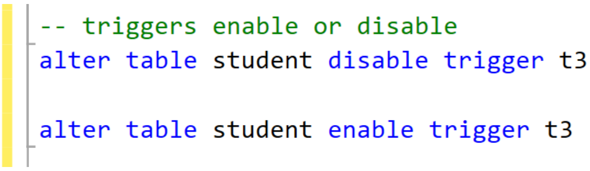


```
-- this triggers not allowed to delet or update on student table
create trigger t2
on student
instead of delete , update
as
    select 'Not allowed for user = ' + suser_name()

update student
    set St_Fname = 'Mahmoud'
    where st_id =123
```

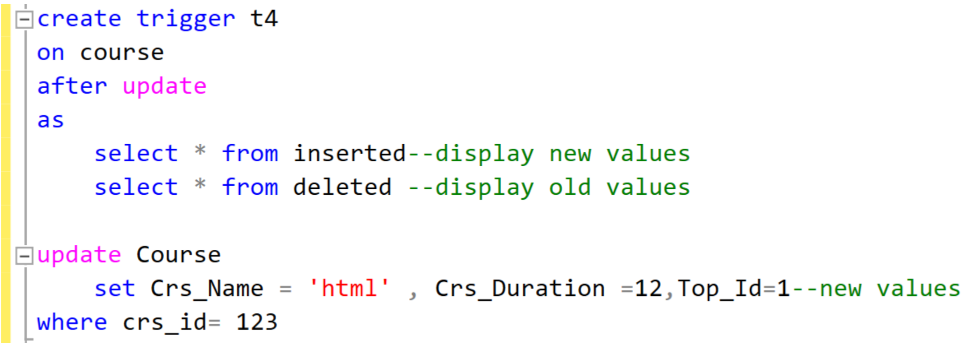
trigger handles both DELETE and INSERT operations on a table



```
-- trigger can applied on specific column
create trigger t3
on student
instead of update
as
    if update(st_fname)
        select 'Not allowed update first name for user = ' + suser_name()

update student
    set St_Fname = 'Mohamed' --Not allowed
```

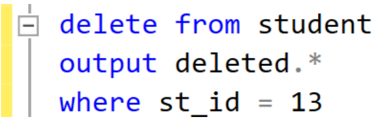


```
-- triggers enable or disable
alter table student disable trigger t3

alter table student enable trigger t3
```



```
create trigger t4
on course
after update
as
    select * from inserted--display new values
    select * from deleted --display old values

update Course
    set Crs_Name = 'html' , Crs_Duration =12,Top_Id=1--new values
where crs_id= 123
```

Output keyword (Triggers on Run time)

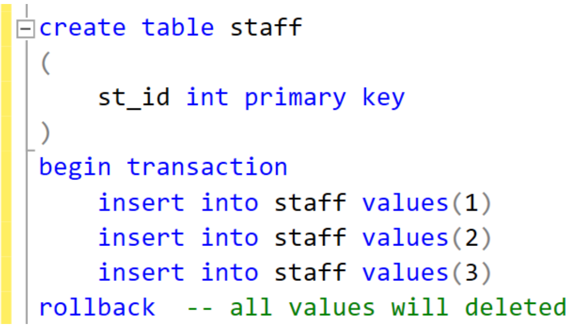


```
delete from student
output deleted.*
where st_id = 13
```

Transaction#:

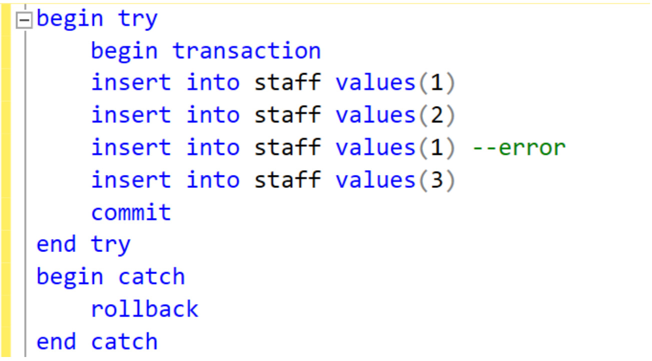
Batch: refers to a group of items, tasks, or operations processed together as a single unit (separated query).

Transaction: a sequence of one or more SQL operations (such as INSERT, UPDATE, DELETE, etc.) that are executed as a single logical unit of work.



```
create table staff
(
    st_id int primary key
)
begin transaction
    insert into staff values(1)
    insert into staff values(2)
    insert into staff values(3)
rollback  -- all values will deleted
```

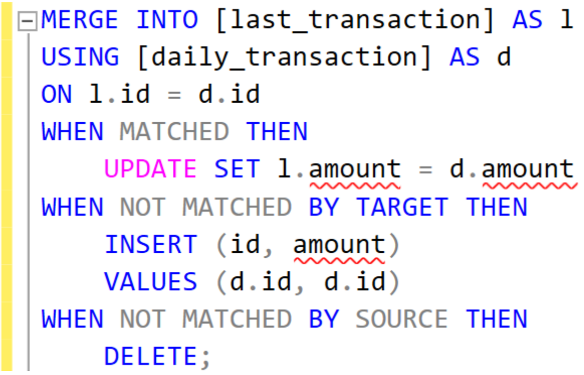
Handle error:



```
begin try
    begin transaction
    insert into staff values(1)
    insert into staff values(2)
    insert into staff values(1) --error
    insert into staff values(3)
    commit
end try
begin catch
    rollback
end catch
```

Merge #:

Definition: statement in SQL is used to perform a combination of INSERT, UPDATE, and DELETE operations in a single query based on certain conditions. It is especially useful for synchronizing two tables or datasets by matching records and performing appropriate actions depending on whether the records exist in one or both datasets.



```sql
MERGE INTO [last_transaction] AS l
USING [daily_transaction] AS d
ON l.id = d.id
WHEN MATCHED THEN
    UPDATE SET l.amount = d.amount
WHEN NOT MATCHED BY TARGET THEN
    INSERT (id, amount)
    VALUES (d.id, d.id)
WHEN NOT MATCHED BY SOURCE THEN
    DELETE;
```

Index #:

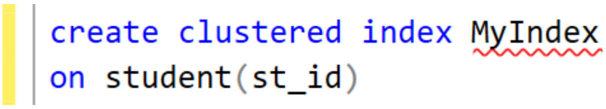
Path: Databases / database_name / table_name / Indexes

Clustered Index: Determines the physical order of rows in the table. A table with a clustered index is called a clustered table.

Data Storage: Data rows are stored directly in the order of the clustered index key.

Number per Table: Only one clustered index per table (since the table can be physically sorted only in one way).

Usage: Best for queries that retrieve ranges of data or frequently sort the data.



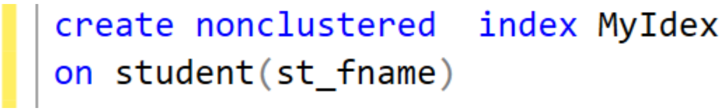
```sql
create clustered index MyIndex
on student(st_id)
```

Non-Clustered Index: Maintains a separate structure from the table for the index. The data rows are not stored in the index order but in the default heap or clustered index order.

Data Storage: Stores only the indexed columns and includes a pointer (Row ID or Clustered Key) to the actual data row.

Number per Table: A table can have multiple non-clustered indexes.

Usage: Best for queries that search for specific columns frequently or need lookups on non-primary key columns.



```sql
create nonclustered  index MyIdex
on student(st_fname)
```

System database #:

1- Master Database: 1-Stores information about the SQL Server instance configuration.

2- Maintains metadata for all other databases on the server (e.g., database locations, logins, system settings).

2- Model Database: 1- Acts as a template for creating new databases.

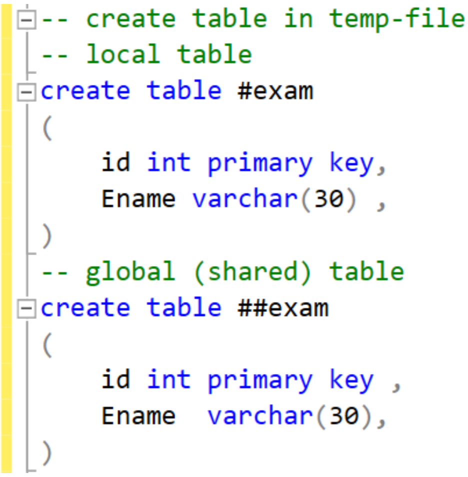
2- Any objects, settings, or configurations in the model database are inherited by newly created databases.

3- msdb Database: 1-Stores information for SQL Server Agent operations.

2- Supports scheduling, jobs, alerts, and backups.

4- Tempdb Database: 1- A temporary workspace used by SQL Server.

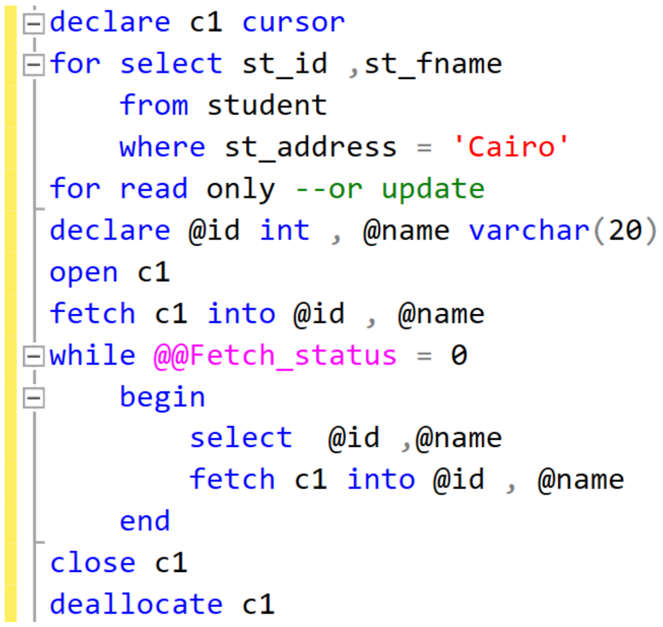
2- Stores temporary tables, variables, and int ermediate results



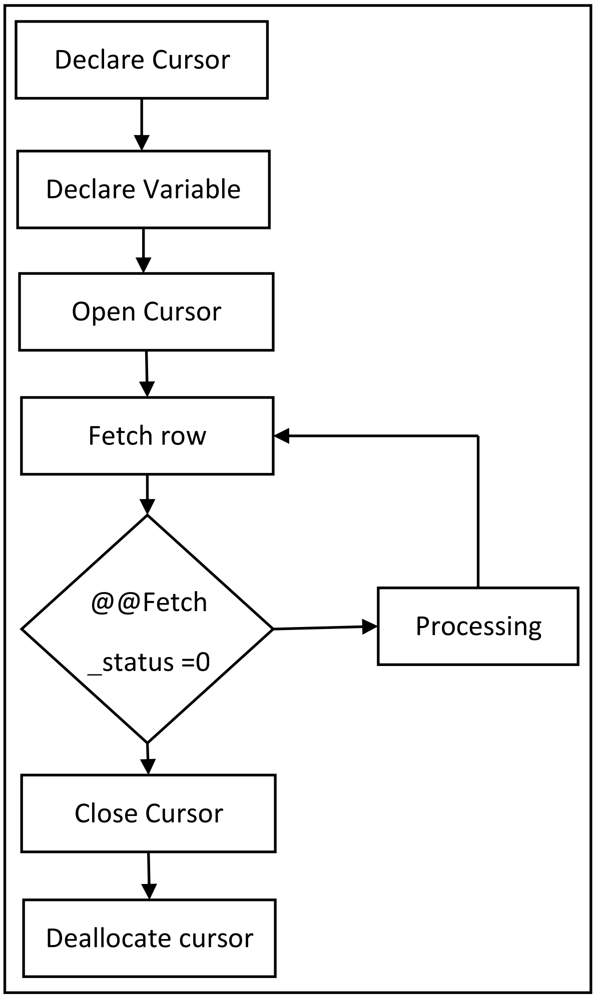
```sql
-- create table in temp-file
-- local table
create table #exam
(
    id int primary key,
    Ename varchar(30) ,
)
-- global (shared) table
create table ##exam
(
    id int primary key ,
    Ename  varchar(30),
)
```

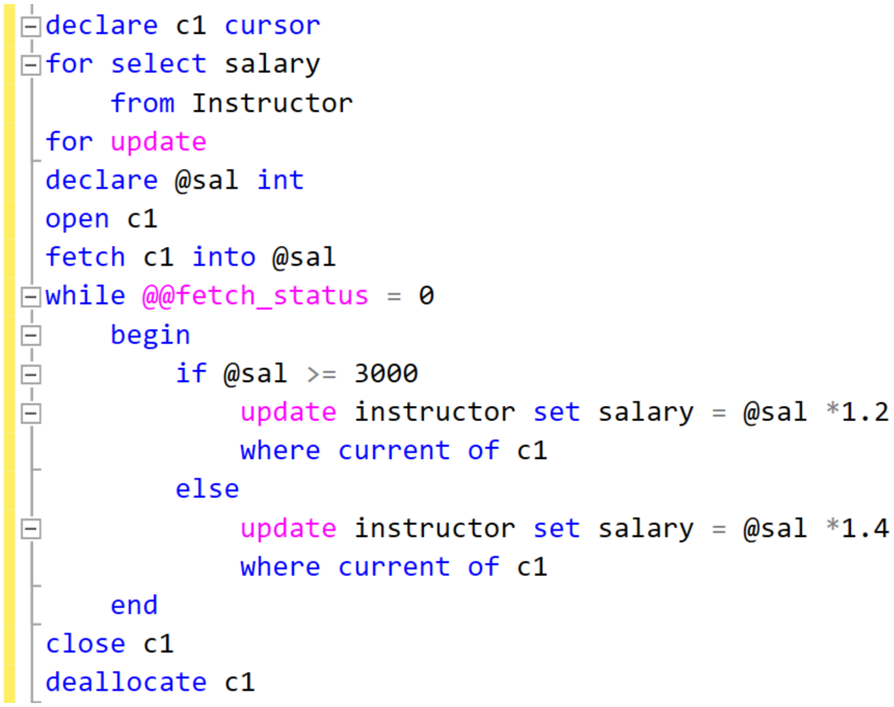
Cursor #:

Definition: used to retrieve and manipulate rows returned by a query one at a time. Cursors are often used when row-by-row processing is required, which cannot be achieved with standard SQL set operations.



```sql
declare c1 cursor
for select st_id ,st_fname
    from student
    where st_address = 'Cairo'
for read only --or update
declare @id int , @name varchar(20)
open c1
fetch c1 into @id , @name
while @@Fetch_status = 0
    begin
        select  @id ,@name
        fetch c1 into @id , @name
    end
close c1
deallocate c1
```





```sql
declare c1 cursor
for select salary
    from Instructor
for update
declare @sal int
open c1
fetch c1 into @sal
while @@fetch_status = 0
    begin
        if @sal >= 3000
            update instructor set salary = @sal *1.2
            where current of c1
        else
            update instructor set salary = @sal *1.4
            where current of c1
    end
close c1
deallocate c1
```

Backup #:

Definition: essential task to ensure data safety and recovery in case of hardware failure, corruption, or other issues. SQL Server provides several backup options to suit different needs.

Types of Backups in SQL Server:

Full Backup: Backs up the entire database.

Differential Backup: Backs up only the data that has changed since the last full backup.

Transaction Log Backup: Backs up the transaction log to ensure point-in-time recovery