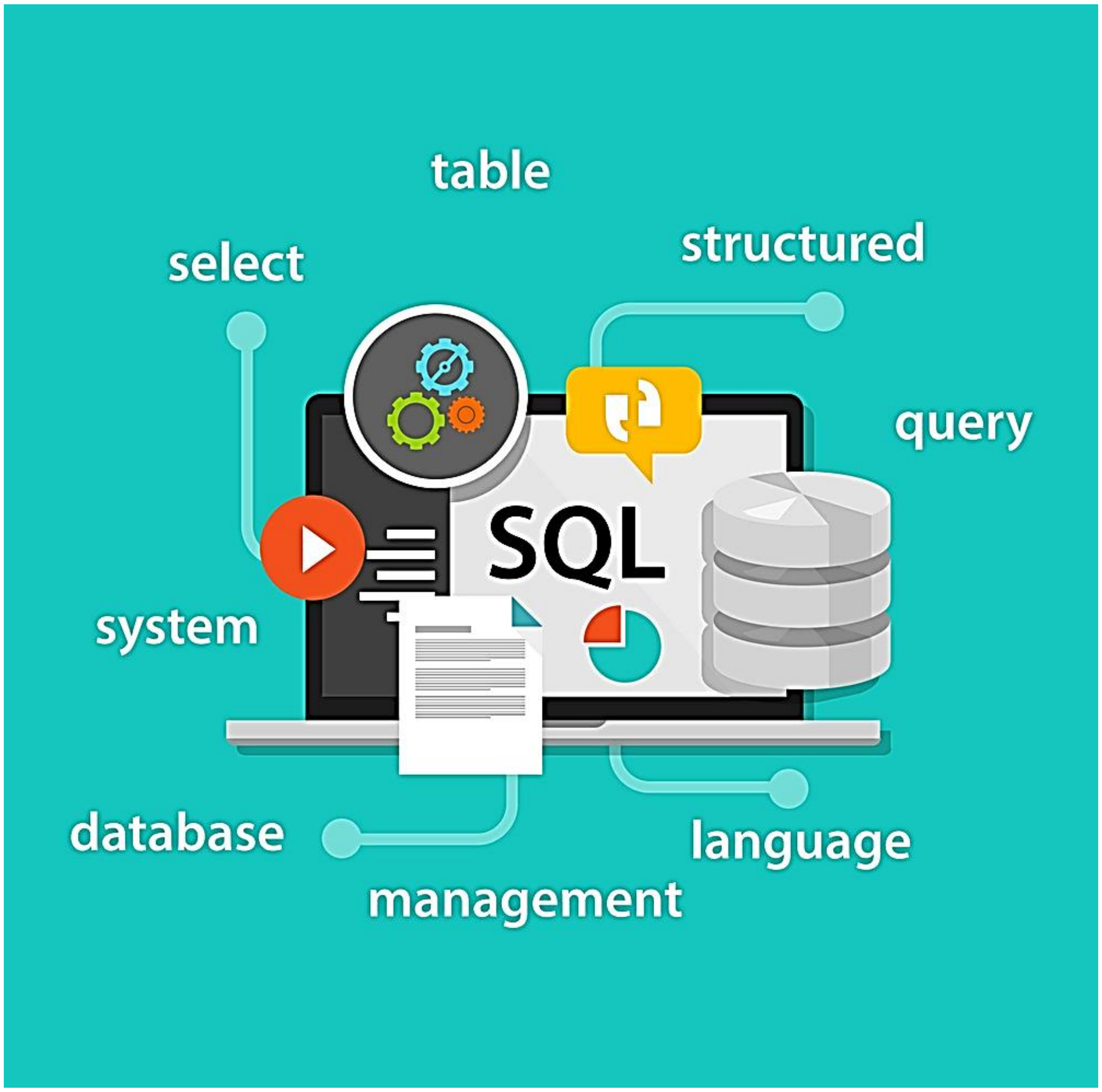


(SQL)

Structured Query Language

Omar Inam Mohammed

SQL (structured query language)

Structured Query Language (SQL) is a database language designed for managing data in a relational database management system. SQL was initially developed by IBM in the early 1970s

SQL components:

DML

DML is abbreviation of *Data Manipulation Language*. used to retrieve, store, modify, delete, insert and update data in tables.

Examples: SELECT, UPDATE, and INSERT statements

DDL

DDL is abbreviation of *Data Definition Language*. used to create and modify the structure of database objects in database.

Examples: CREATE, ALTER, DROP statements

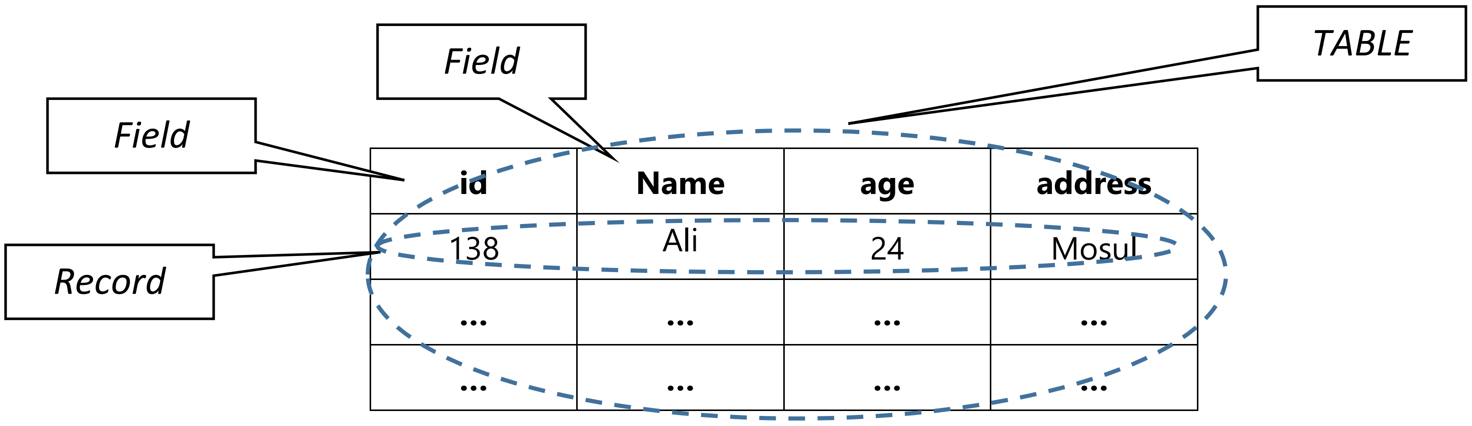
DCL

DCL is abbreviation of *Data Control Language*. used to create roles, permissions, and referential integrity as well it is used to control access to database by securing it.

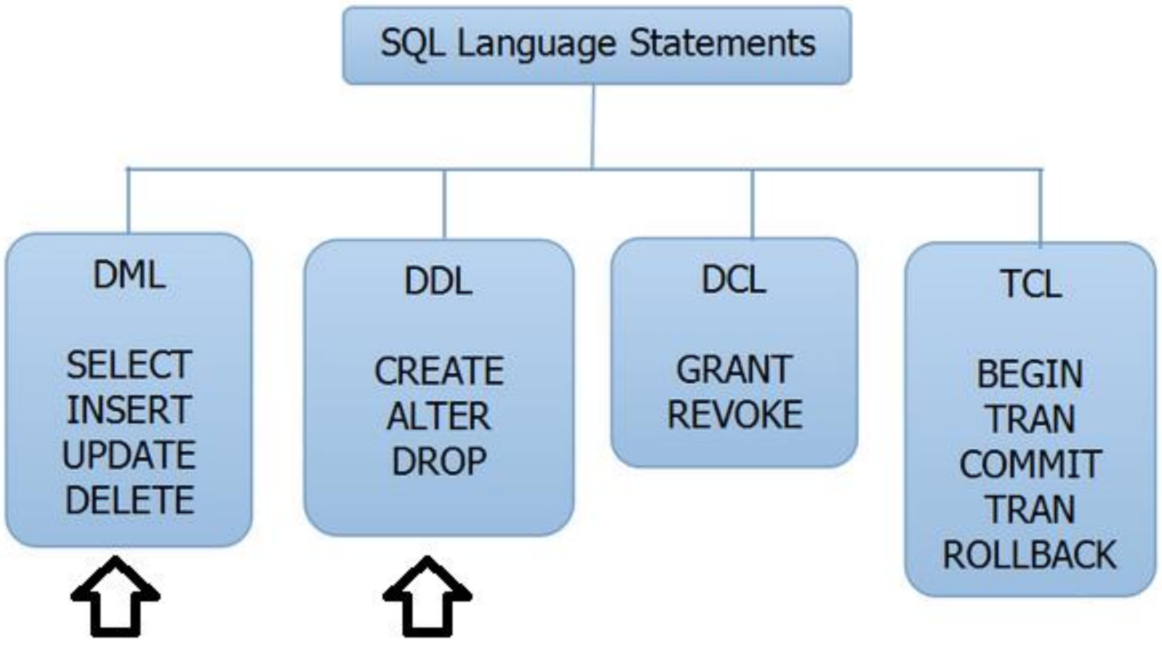
Examples: GRANT, REVOKE statements

TCL

TCL is abbreviation of *Transactional Control Language*. used to manage different transactions occurring within a database.



id	Name	age	address
138	Ali	24	Mosul
...	...	...	...
...	...	...	...



DataType: The data type, as described below, must be a system data type or a user-defined data type. Many of the data types have a
size such as CHAR(35).

• int	for example : 5 , 568 , -78154 …
• text (size) or char(size)	for example : 'computer' , 'ali' 'AS5'
• date	for example : #2/6/1992#
• float	for example : 56.387 , 0.256 , 564.08
• autoincrement	for example : 1,2,3,4,5,6,7,8,9 …

- **DDL:**

Create table syntax: used to create tables

CREATE TABLE *table_name* (*column1 datatype*, *column2 datatype*, *column3 datatype*, ...);

Example:
Create table person (name char (40), address char (30), age int, average float, inroll_date date);

Alter table syntax: used to add or remove columns to table

ALTER TABLE table_name ADD column_name datatype;
ALTER TABLE table_name DROP column_name;

Example:
Alter table student add stage_name text(30)
Alter table student drop address

Drop table syntax : used to delete tables

DROP TABLE table_name

- **DML:**

Insert into syntax: used to add new records to table

```
INSERT INTO table_Name  VALUES ('value_1',' value_2', ......... );
```

Example:

Insert into person values (‘ahmad’ , ‘mosul’ , 28 , 67.35 , #2/5/2015#)

Hints :

Values must equal number of fields.

Values must arrange in order of fields

Update syntax: used to change exist records values

```
UPDATE table_name SET column_name = new_value WHERE column_name =
value ;
```

Example:

Update student set name=‘Basil’ , city=‘Baghdad’ where id = 294

Delete syntax: use to delete records from tables

```
DELETE FROM table_name WHERE column_name = value ;
```

Example:

Delete from student where name=‘mazin’

- ❖ WHERE clause like condition expression in other programming language
- ❖ We can use with this clause mathematical operation : (= , > , < , >= , <= , <>)

EXAPMLE: you have the following table header (fields names) given below:

PATIENT

P_ID	***P_Name***	***P_Age***	***P_Case***	***Entry_Date***	***Bed_Number***
int	char	int	Char	date	Int
…….	…………	…….	……………	………….	…………

❖ By Using **SQL** , Do the following :

Q1. Add new record with random values.

```
insert into patient values(5,'Ali',22,'Anemia',(3/2/2016),55)
```

Q2. Delete field ***P_Age*** from PATIENT .

```
alter table patient drop P_Age
```

Q3. Add field ***exit_date*** (*char*) to PATIENT .

```
alter table patient add exit_date char(10)
```

Q4. Delete record ***P_ID*** = 113 .

```
delete from patient where P_ID = 113
```

Q5. Delete table ***PATIENT***.

```
drop table patient
```

SELECT (*Idea of Query*):

SQL ***select*** statement retrieves records from tables based on clauses that specify criteria.
Select syntax:

```
SELECT * FROM table_name WHERE .......
```

Examples:

1- Select * from student (retrieve all records in specific table)
2- Select name , age from person
3- Select DISTINCT * from worker (delete repetition)

Select with ***WHERE*** criteria
With (WHERE) we can use logical statement and logical operation
Example:

1- Select name, age from employee where id**=**13
2- Select * from student where average **>=** 50
3- Select ID, address from worker where city**=**'Mosul' ***AND*** age **<** 22
4- Select * from order where type **!=**'computer'

Select with ***IN*** criteria
With (IN) we can determine specific values
Example:

1- Select name, age from employee where name IN ('Ali' , 'Ahmad' , 'Samir)
2- Select * from student where average IN (90 , 80, 70)

Select with ***BETWEEN AND*** criteria
This clause give us ranges
Example:

1- Select * from student where average BETWEEN 50 AND 60

Select with ***LIKE*** criteria

LIKE can be used to find the pattern by using **(*) , (?) , (#)**

(*) refers to random number of characters

(?) refers to one character

(#) refers to digit (numeric character)

(not) use with like

(trim) use with column data type text when you use like

Example:

1- Select * from student where name LIKE('a *') // fields start with **a**
2- Select * from student where name LIKE('w ???') // fields start with **w** and **3** characters
3- Select * from student where name LIKE('B * S') // fields start with **B** and end with **S**
4- Select * from student where name LIKE('*[a-d]') // fields end with range [**a.b.c.d**]

Select with ***ORDER BY (DESC & ASCE)*** criteria

This clause arrange depending on character ASCII code

Example:

1- Select * from student ORDER BY name DESC

Select with ***arithmetic operation***

It is possible to use arithmetic operation to retrieve some new columns.

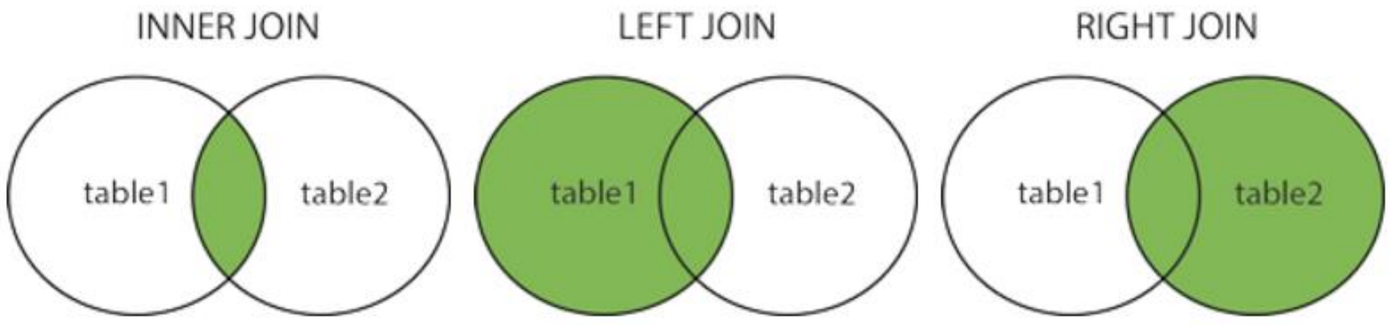
Example:

1- Select name, salary *0.1 as rate, salary+1000 from employee
2- Select ID, Avg+5 as new_av from student

JOIN clause

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Different types of JOINs

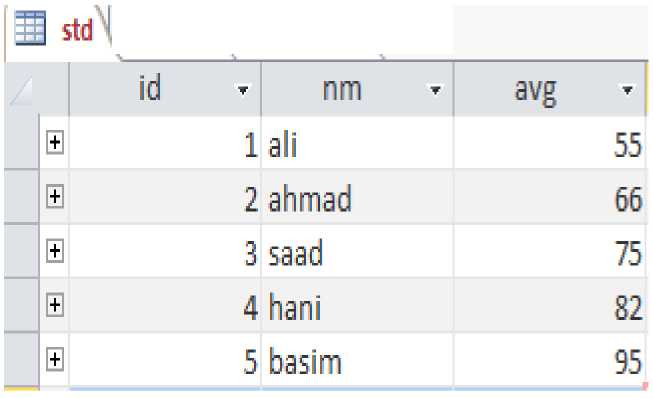


1. **INNER JOIN**: returns rows when there is a match in both tables.
2. **LEFT OUTER JOIN**: returns all rows from the left table, even if there are no matches in the right table.
3. **RIGHT OUTER JOIN**: returns all rows from the right table, even if there are no matches in the left table

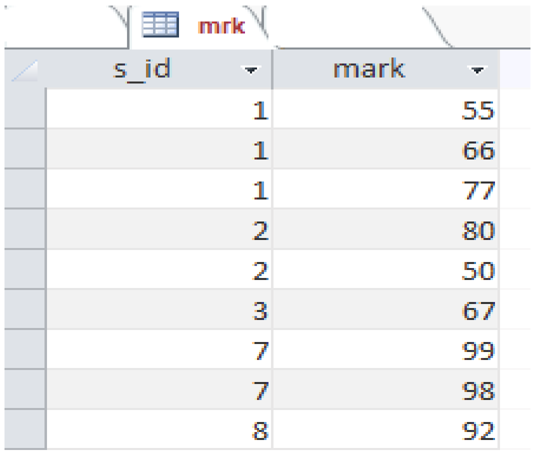
1- (**INNER**) JOIN:

```
SELECT column-names
FROM table-name1 INNER JOIN table-name2
 ON column-name1 = column-name2
 WHERE condition
```

Example:


std

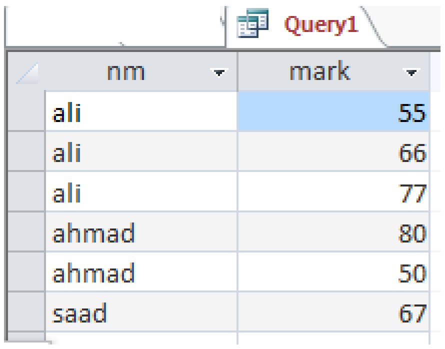
id	nm	avg
1	ali	55
2	ahmad	66
3	saad	75
4	hani	82
5	basim	95


mrk

s_id	mark
1	55
1	66
1	77
2	80
2	50
3	67
7	99
7	98
8	92

SELECT std.nm, mrk.mark
FROM std INNER JOIN mrk ON std.id = mrk.s_id;

Result:


Query1

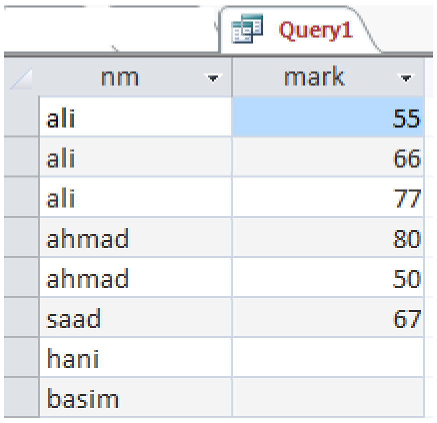
nm	mark
ali	55
ali	66
ali	77
ahmad	80
ahmad	50
saad	67

2- **LEFT** (OUTER) JOIN:

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2 ON table1.column_name = table2.column_name;
```

Example:
SELECT std.nm, mrk.mark
FROM std LEFT JOIN mrk ON std.id = mrk.s_id;

Result:



nm	mark
ali	55
ali	66
ali	77
ahmad	80
ahmad	50
saad	67
hani	
basim	

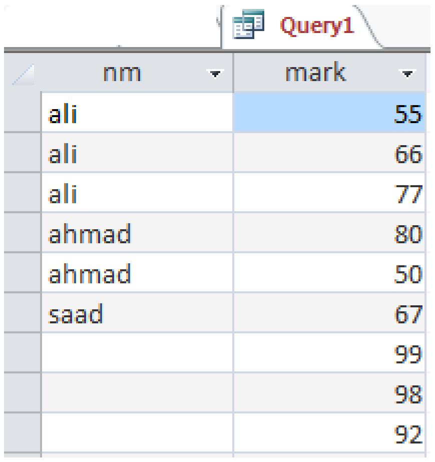
3- **RIGHT** (OUTER) JOIN:

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2 ON table1.column_name = table2.column_name;
```

Example:

SELECT std.nm, mrk.mark
FROM std RIGHT JOIN mrk ON std.id = mrk.s_id;

Result:



Query1

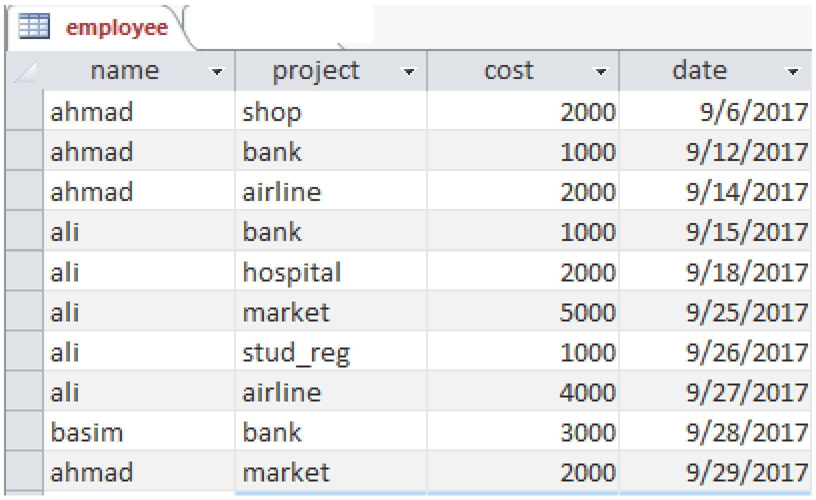
nm	mark
ali	55
ali	66
ali	77
ahmad	80
ahmad	50
saad	67
	99
	98
	92

GROUP BY WITH SELECT:

The GROUP BY statement is often used with aggregate functions (**COUNT**, **MAX**, **MIN**, **SUM**, **AVG**) to group the result-set by one or more columns.

SELECT column(s) , ***function***(column_name(s))
FROM table_name
WHERE condition
GROUP BY column_name(s)

Example:



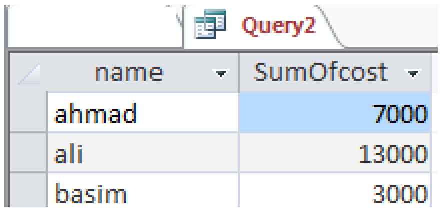
employee

name	project	cost	date
ahmad	shop	2000	9/6/2017
ahmad	bank	1000	9/12/2017
ahmad	airline	2000	9/14/2017
ali	bank	1000	9/15/2017
ali	hospital	2000	9/18/2017
ali	market	5000	9/25/2017
ali	stud_reg	1000	9/26/2017
ali	airline	4000	9/27/2017
basim	bank	3000	9/28/2017
ahmad	market	2000	9/29/2017

# 1:

SELECT employee.name, **SUM**(employee.cost) AS SumOfcost
FROM employee
GROUP BY employee.name;

Result:



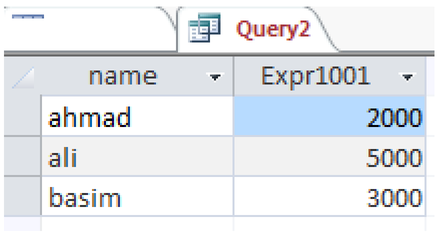
Query2

name	SumOfcost
ahmad	7000
ali	13000
basim	3000

# 2:

SELECT employee.name, **MAX**(employee.cost)
FROM employee
GROUP BY employee.name;

Result:

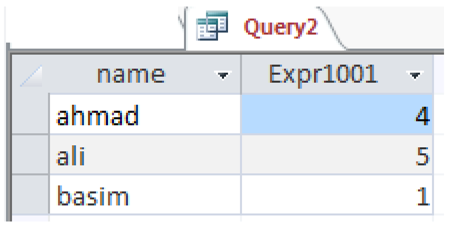

Query2

name	Expr1001
ahmad	2000
ali	5000
basim	3000

# 3:

SELECT employee.name, **COUNT**(employee.cost)
FROM employee
GROUP BY employee.name;

Result:


Query2

name	Expr1001
ahmad	4
ali	5
basim	1

HAVING with GROUP BY:

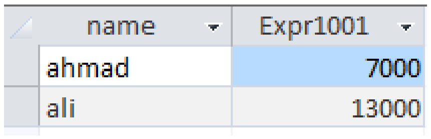
The HAVING Clause enables you to specify conditions that filter which group results appear in the results.
The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
```

Example:

```
SELECT employee.name, sum(employee.cost)
FROM employee
GROUP BY employee.name
having count(employee.name)>3;
```

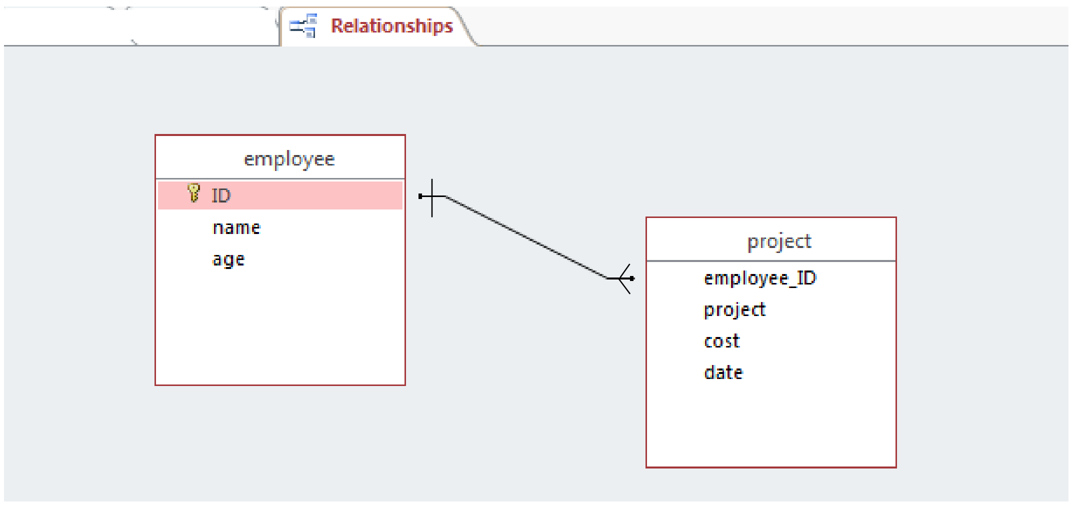
Result:

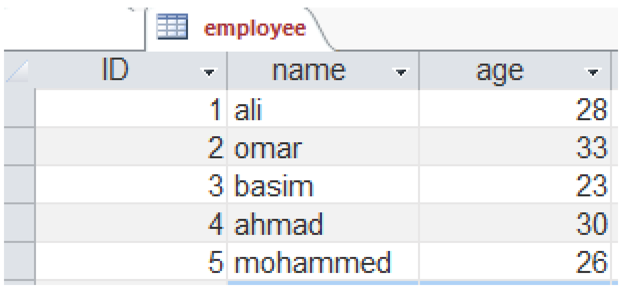


name	Expr1001
ahmad	7000
ali	13000

Exercise:

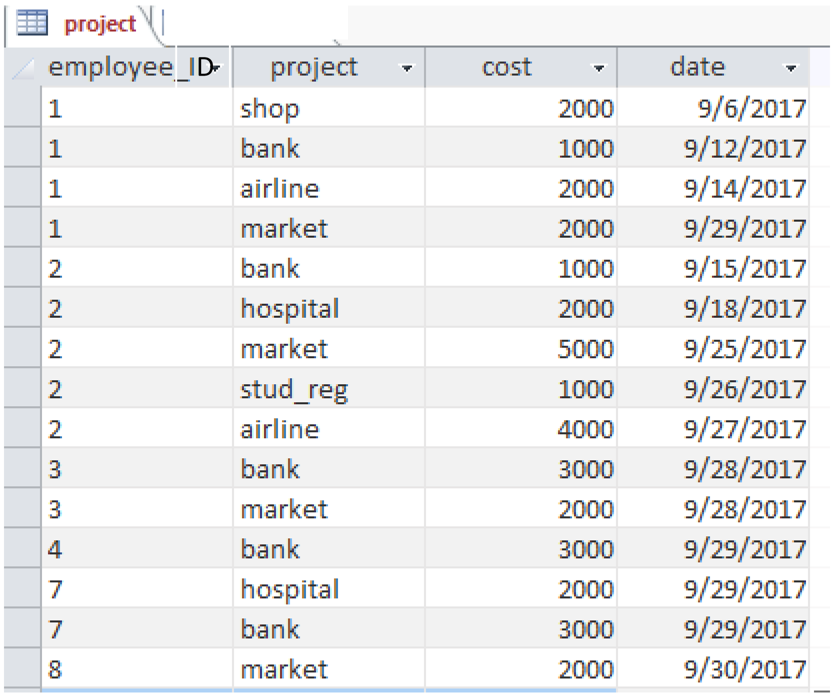
#A -





employee

ID	name	age
1	ali	28
2	omar	33
3	basim	23
4	ahmad	30
5	mohammed	26



project

employee_ID	project	cost	date
1	shop	2000	9/6/2017
1	bank	1000	9/12/2017
1	airline	2000	9/14/2017
1	market	2000	9/29/2017
2	bank	1000	9/15/2017
2	hospital	2000	9/18/2017
2	market	5000	9/25/2017
2	stud_reg	1000	9/26/2017
2	airline	4000	9/27/2017
3	bank	3000	9/28/2017
3	market	2000	9/28/2017
4	bank	3000	9/29/2017
7	hospital	2000	9/29/2017
7	bank	3000	9/29/2017
8	market	2000	9/30/2017

#B- Depending on the tables that were created in previous branch. Write **SQL** Queries to view:

1. employees names whose age more than or equal 28.

```
SELECT employee.name
FROM employee where age >= 28;
```

2. employees ages whose names start wihe ‘a’.

```
SELECT employee.age
FROM employee where name like('a*')
```

3. employees information whose Names contain ‘M’ character and ages are between (20-30)

```
SELECT *
FROM employee where name like ( '*m*' ) AND age between 20 and 30
```

4. employees IDs with count of project for each emplyee.

```
SELECT employee_id,count(project)
FROM project group by employee_id
```

5. alphabetical employee’s names with average cost for each employee has more than 3 projects.

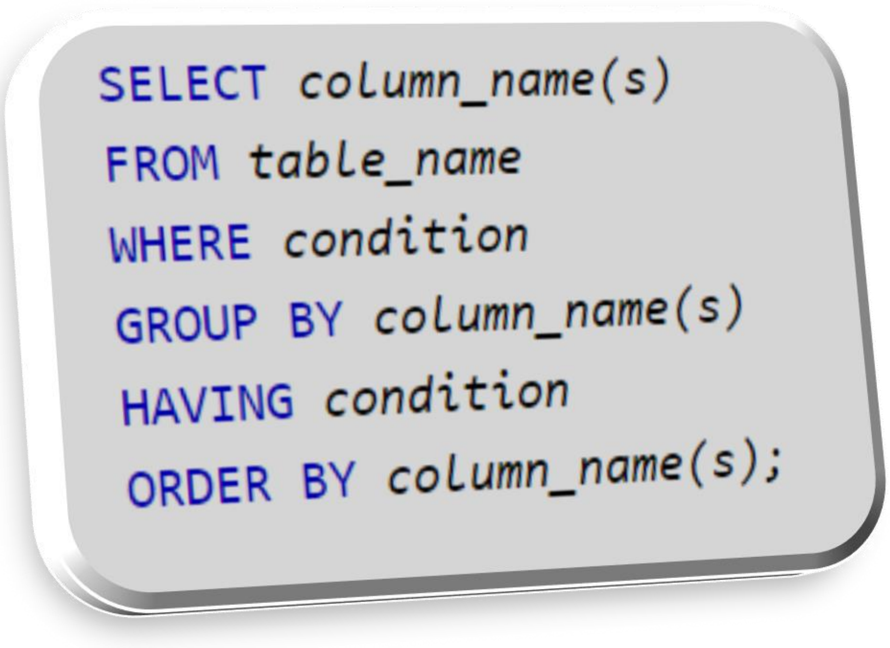
```
SELECT employee.name, Avg(project.cost) AS AvgOfcost
FROM employee INNER JOIN project ON employee.ID = project.employee_ID
GROUP BY employee.name
HAVING Count(project.project)>3
ORDER BY employee.name
```

6. employees names who haven't project and their ages more than 25

```
SELECT employee.name,count( project.project )
FROM employee LEFT JOIN project ON employee.ID = project.employee_ID
where employee.age  >= 25
group by  employee.name
having count(project.project)=0
```

Summery:

SELECT general syntax:



```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

ACCESS Functions

- **String function:**
  1) Left (String , int)
  2) Right(String, int)
  3) Len( )
  4) Ucase( )
  5) Lcase( )
  6) Ltrim( )
- **Numeric Function:**
  1) ABS( )
  2) Round( )
  3) Fix( )
- **Date Function:**
  1) date( )
  2) Day( )
  3) Month( )
  4) Year( )

Example:

1. select upper(name) , ABS(rank) from student where id=3
2. select ucase(city) , round(average) from student where name like ‘ *A* ’

SQL Constraints

SQL constraints are used to specify rules for data in a table.

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

```
❖ CREATE TABLE table_name (
      column1 datatype constraint,
      column2 datatype constraint,
      column3 datatype constraint,
      ....
  );


❖ ALTER TABLE table_name ADD/DROP
  CONSTRAINT constraint _name
```

1. NOT NULL Constraint by default, a column can hold NULL values. The NOT NULL constraint enforces a column to NOT accept NULL values.

Example:

create table employee (id int not null, name char (30) not null)

2. The UNIQUE Constraint ensures that all values in a column are different.

Example:

create table employee (id int NOT NULL, name char (30) UNIQUE)

3. PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL values. A table can have only ONE primary key; and in the table.

Example:

```
CREATE TABLE person (id int CONSTRAINT id1 PRIMARY KEY, ToyID
int NOT NULL)
```

4. FOREIGN KEY Constraint

A FOREIGN KEY is a key used to link two tables together. A FOREIGN KEY is a field in one table that refers to the PRIMARY KEY in another table.

Example:

```
create table mark (Eid int , grade int , CONSTRAINT m1 foreign key
(Eid) references employee (id) )
```

5. PRIMARY KEY Constraint (ALTER TABLE)

Example:

```
alter table student Add CONSTRAINT pk1 PRIMARY KEY (id)
```

6. PRIMARY KEY Constraint (ALTER TABLE)

Example:

```
ALTER TABLE student DROP CONSTRAINT pk1
```

7. FOREIGN KEY Constraint (ALTER TABLE)

Example:

```
ALTER TABLE mark add  CONSTRAINT mfk foreign key (sid)
references std (id) )
```

8. FOREIGN KEY Constraint (ALTER TABLE)

Example:

```
ALTER TABLE mark DROP CONSTRAINT mfk
```

9. ALTER TABLE (change column type)

Example:

```
ALTER TABLE stud ALTER COLUMN nm int
```

Sub Queries

What is subquery in SQL?

A subquery is a SQL query nested inside a larger query.

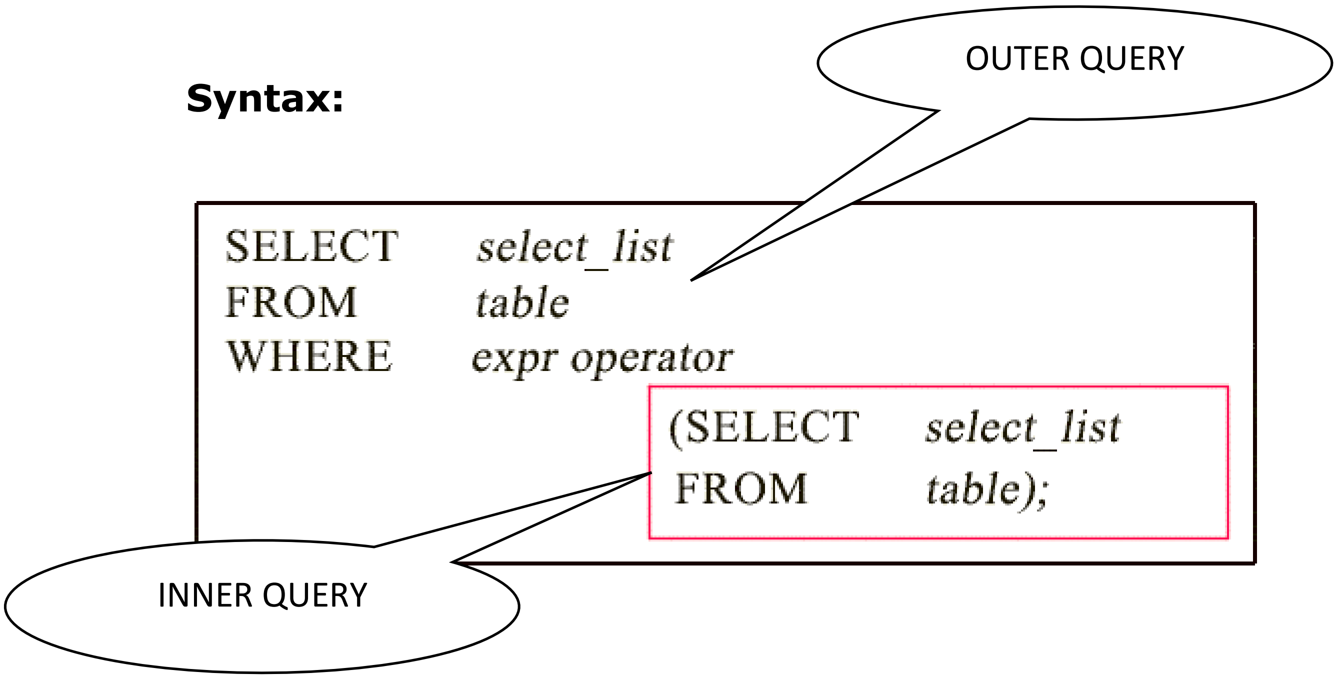
The subquery can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another subquery.

A subquery is usually added within the WHERE Clause of another SQL SELECT statement.

A subquery is also called an inner query or inner select, while the statement containing a subquery is also called an outer query or outer select.

The inner query executes first before its parent query so that the results of an inner query can be passed to the outer query.

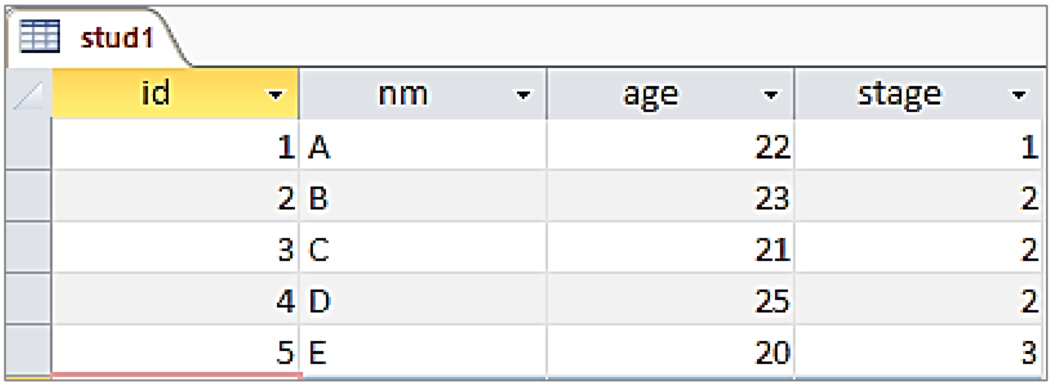
Syntax:



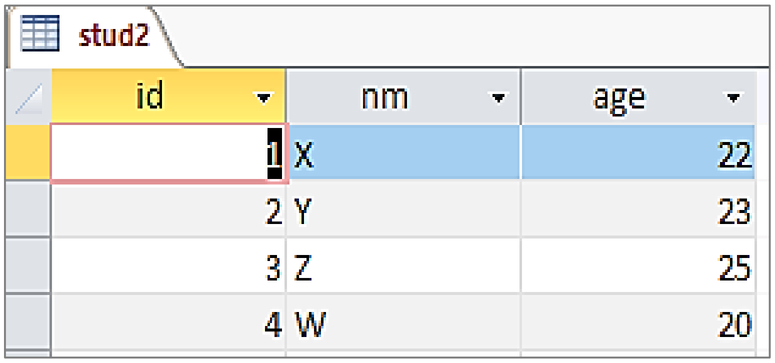
```
SELECT     select_list
FROM       table
WHERE      expr operator
                        (SELECT    select_list
                         FROM      table);
```

Types of Sub Queries:

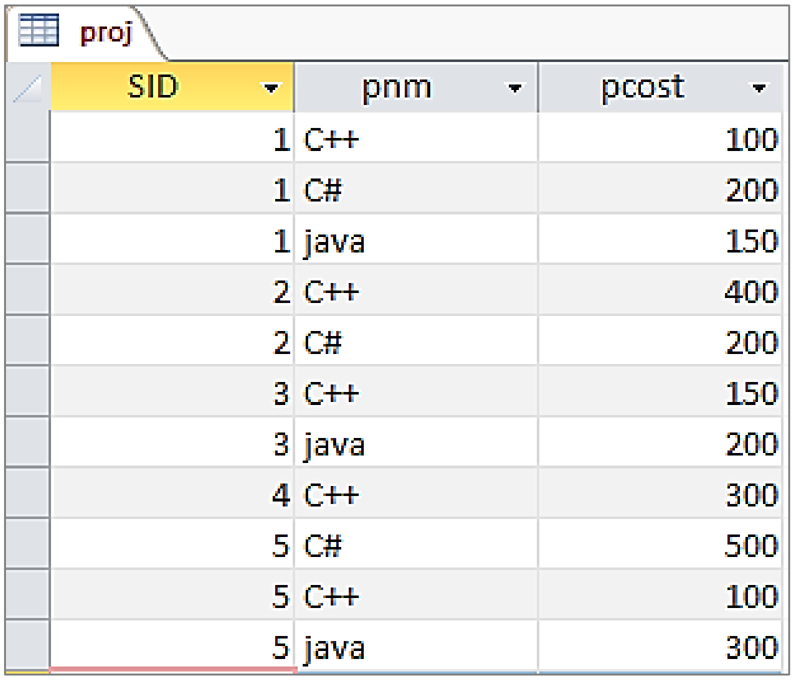
- **Single Row Sub Query**: Sub query which returns single row output. We can use **WHERE** and **(=, <>,>, <, <=,>=)** operations.
- **Multiple Row Sub Query**: Sub query returning multiple row output. They make use of multiple row comparison operators **IN, ANY, ALL, NOT**. There can be sub queries returning multiple columns also.
- **Correlated Sub Query**: Correlated subqueries depend on data provided by the outer query. This type of subquery also includes subqueries that use the **EXISTS** operator to test the existence of data rows satisfying specified criteria.


stud1

id	nm	age	stage
1	A	22	1
2	B	23	2
3	C	21	2
4	D	25	2
5	E	20	3

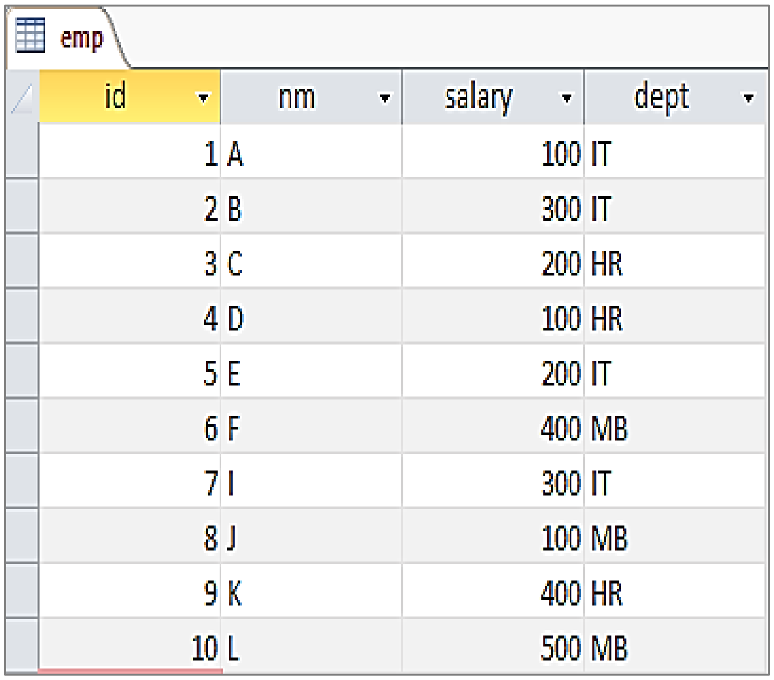

stud2

id	nm	age
1	X	22
2	Y	23
3	Z	25
4	W	20



proj

SID	pnm	pcost
1	C++	100
1	C#	200
1	java	150
2	C++	400
2	C#	200
3	C++	150
3	java	200
4	C++	300
5	C#	500
5	C++	100
5	java	300



emp

id	nm	salary	dept
1	A	100	IT
2	B	300	IT
3	C	200	HR
4	D	100	HR
5	E	200	IT
6	F	400	MB
7	I	300	IT
8	J	100	MB
9	K	400	HR
10	L	500	MB

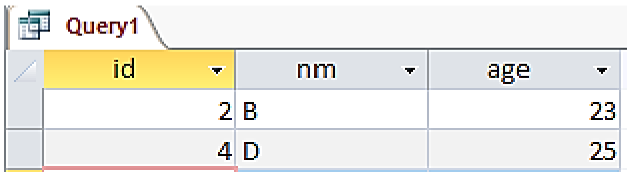
Single Row Sub Query:

A single row subquery returns zero or one row to the outer SQL statement.

Example: *give the info of all students in stud1 who's bigger than 'Y' in stud2*

```
SELECT id,nm,age FROM stud1
WHERE age >= (select age from stud2 where nm='Y');
```

Result:



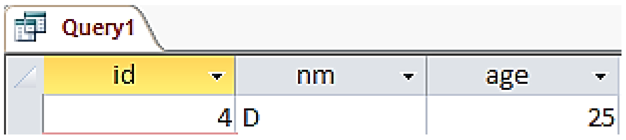
id	nm	age
2	B	23
4	D	25

Example: *give the info of oldest student*

SELECT id,nm,age FROM stud1

WHERE age = (select max(age) from stud1)

Result:



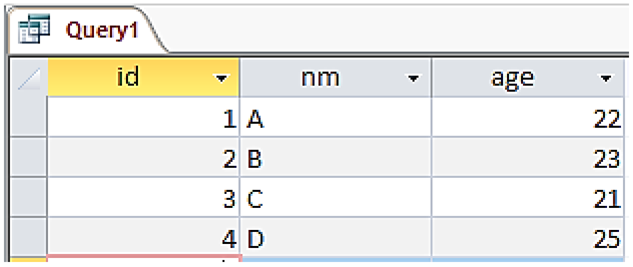
id	nm	age
4	D	25

Example: *hide the info of smallest student*

SELECT id,nm,age FROM stud1

WHERE age > (select MIN(age) from stud1)

Result:



Query1

id	nm	age
1	A	22
2	B	23
3	C	21
4	D	25

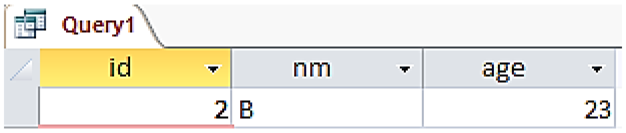
Example: *give the info of second biggest student*

SELECT id,nm,age FROM stud1

WHERE age = (SELECT max(age) FROM stud1

WHERE age < (select max(age) from stud1));

Result:



Query1

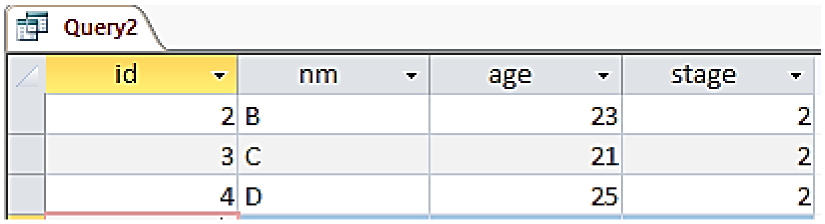
id	nm	age
2	B	23

Example: *give the info of all students in '**B**' student's Stage*

SELECT * FROM stud1

WHERE stage = (SELECT stage FROM stud1 WHERE nm='B')

Result:



id	nm	age	stage
2	B	23	2
3	C	21	2
4	D	25	2

Multiple row sub query

Multiple row subquery returns one or more rows to the outer SQL statement. You may use the **IN , ANY, or ALL** operator in outer query to handle a subquery that returns multiple rows.

- **IN** operator is used to checking a value within a set of values. The list of values may come from the results returned by a subquery
- **ANY** operator to compare a value with any value in a list.
- **ALL** operator to compare a value with all value in a list.

❖ With **ALL** and **ANY** we must use **(> , < , <= , >=)**

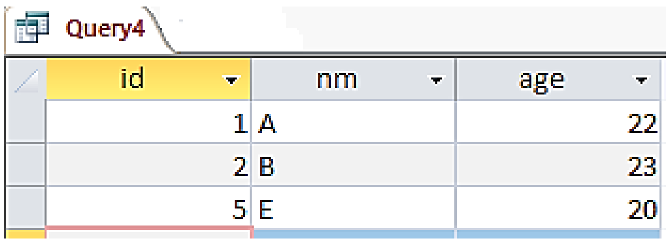
--

Example: *give the info of all students who have C# project*

SELECT id, nm, age FROM stud1

WHERE id IN (select sid from proj where pnm='C#')

Result:


Query4

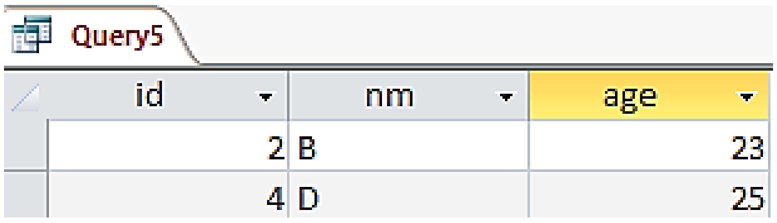
id	nm	age
1	A	22
2	B	23
5	E	20

Example: *give the info of all students who haven't java project*

SELECT id, nm, age FROM stud1

WHERE id NOT IN (select sid from proj where pnm='java')

Result:


Query5

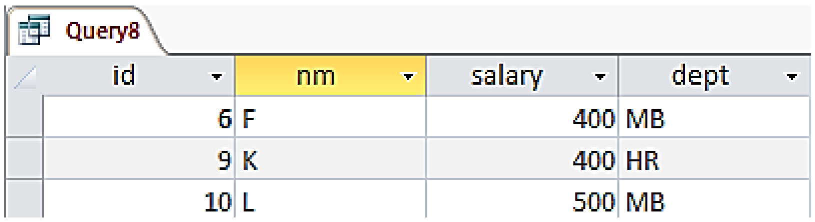
id	nm	age
2	B	23
4	D	25

Example: *give the info of all emp whos salary more than all emp work in dept 'IT'*

SELECT * FROM emp

WHERE salary > ALL (select salary from emp where dept='IT');

Result:



Query8

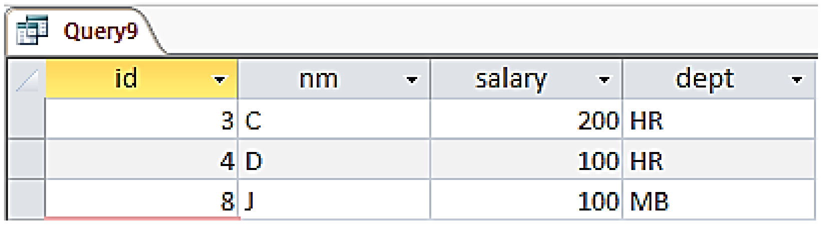
id	nm	salary	dept
6	F	400	MB
9	K	400	HR
10	L	500	MB

Example: *give the info of all emp whos salary less than any emp work in dept 'IT'*

SELECT * FROM emp

WHERE dept<>'IT' AND salary < any (select salary from emp where dept='IT');

Result:



Query9

id	nm	salary	dept
3	C	200	HR
4	D	100	HR
8	J	100	MB

Correlated Sub Query

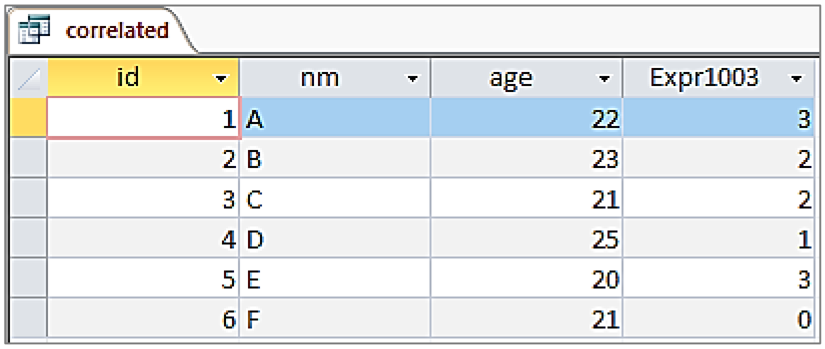
In a SQL database query, a correlated subquery (also known as a synchronized subquery) is a subquery (a query nested inside another query) that uses values from the outer query. Because the subquery may be evaluated once for each row processed by the outer query.

Example: *give the info of all stud1 and the count of proj for each one*

SELECT id, nm, age, (select count(pnm) from proj

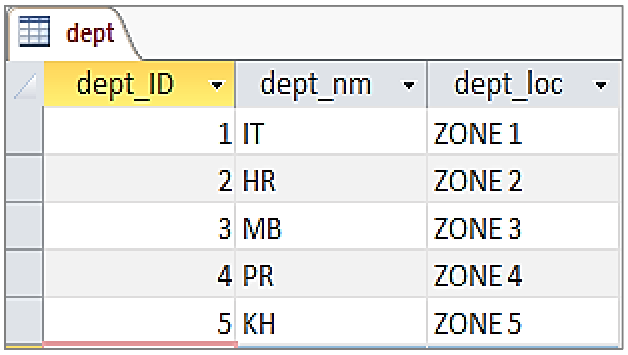
where proj.sid=stud1.id) FROM stud1

Result:



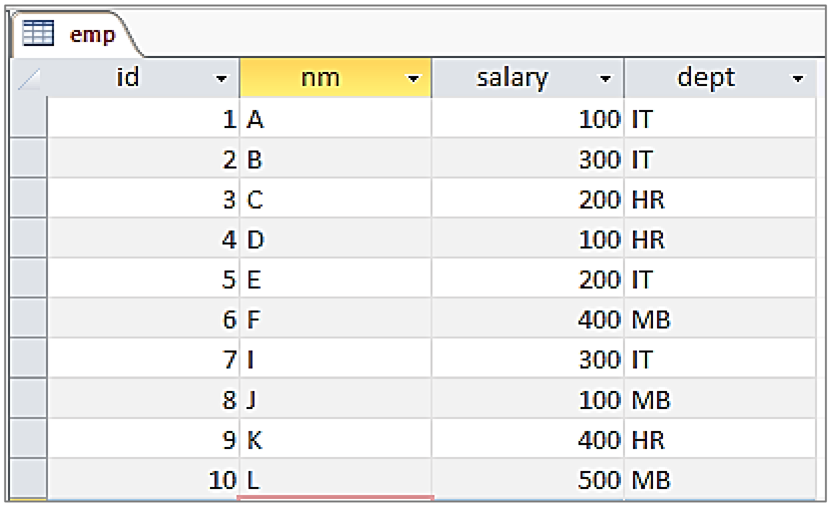
correlated

id	nm	age	Expr1003
1	A	22	3
2	B	23	2
3	C	21	2
4	D	25	1
5	E	20	3
6	F	21	0



dept

dept_ID	dept_nm	dept_loc
1	IT	ZONE 1
2	HR	ZONE 2
3	MB	ZONE 3
4	PR	ZONE 4
5	KH	ZONE 5



emp

id	nm	salary	dept
1	A	100	IT
2	B	300	IT
3	C	200	HR
4	D	100	HR
5	E	200	IT
6	F	400	MB
7	I	300	IT
8	J	100	MB
9	K	400	HR
10	L	500	MB

EXISTS & NOT EXSITS

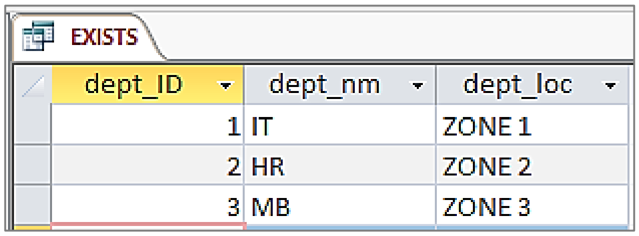
Example: *EXISTS*

SELECT * FROM dept

WHERE EXISTS (select * from emp

where dept.dept_nm=emp.dept)

Result:



EXISTS

dept_ID	dept_nm	dept_loc
1	IT	ZONE 1
2	HR	ZONE 2
3	MB	ZONE 3

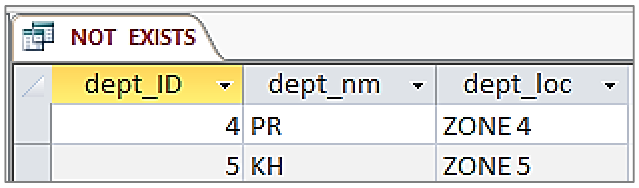
Example: *NOT EXISTS*

SELECT * FROM dept

WHERE NOT EXISTS (select * from emp

where dept.dept_nm=emp.dept)

Result:



NOT EXISTS

dept_ID	dept_nm	dept_loc
4	PR	ZONE 4
5	KH	ZONE 5
