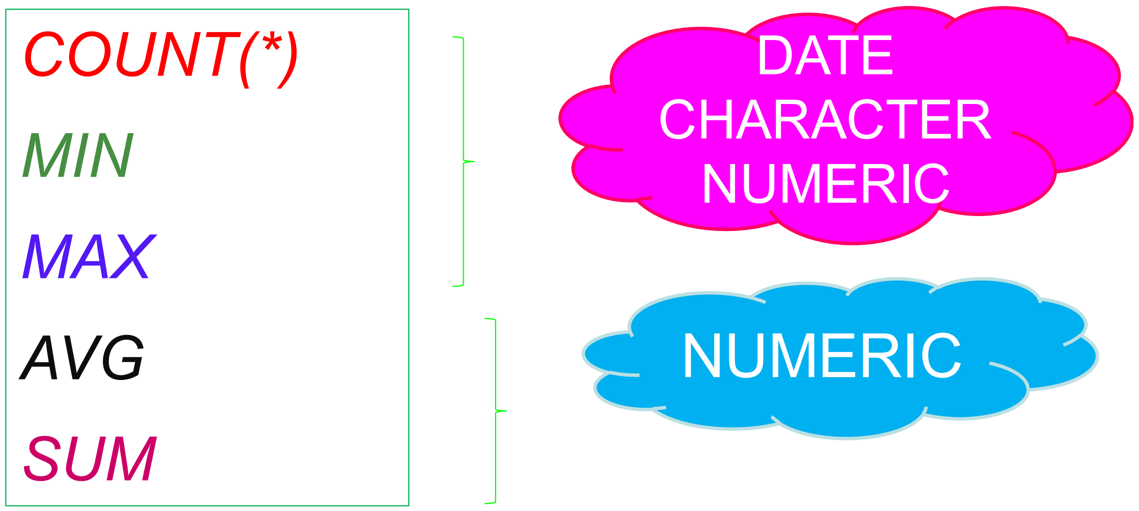
AGGREGATE FUNCTIONS

Mysql also supports group(aggregate) functions like other RDBMS. Let us say we want to find employees working in deptno 20.

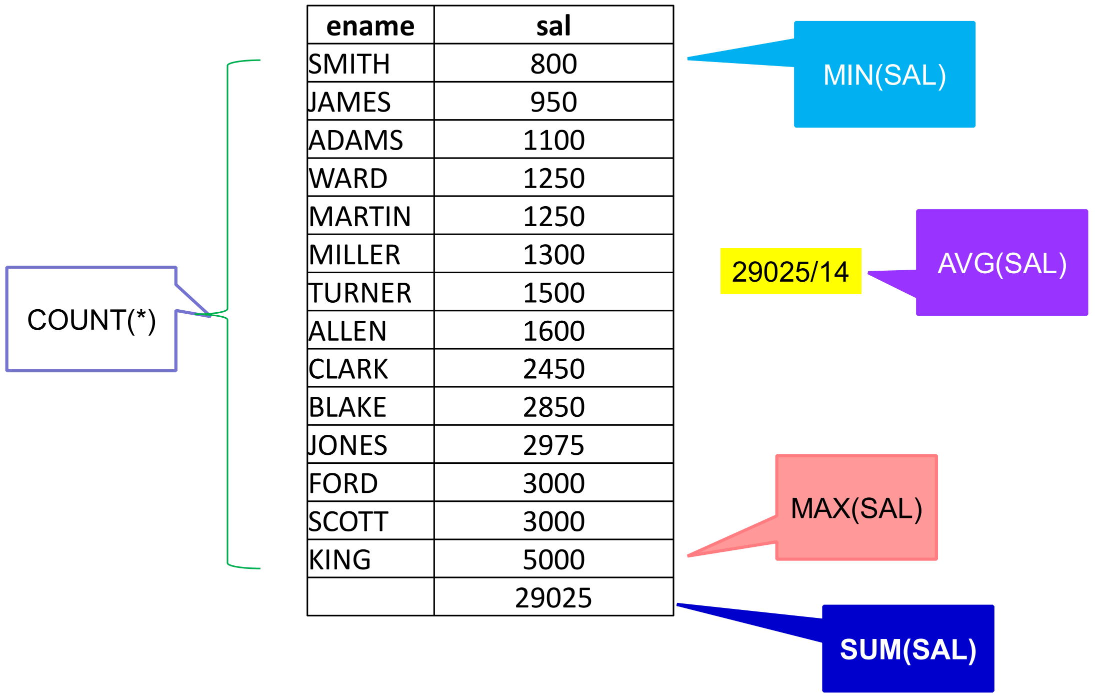
mysql> select * from emp where deptno=20;

```
+-------+-------+---------+------+------------+------+------+--------+
| EMPNO | ENAME | JOB     | MGR  | HIREDATE   | SAL  | COMM | DEPTNO |
+-------+-------+---------+------+------------+------+------+--------+
|  7369 | SMITH | CLERK   | 7902 | 1980-12-17 |  800 | NULL |     20 |
|  7566 | JONES | MANAGER | 7839 | 1981-04-02 | 2975 | NULL |     20 |
|  7788 | SCOTT | ANALYST | 7566 | 1987-07-13 | 3000 | NULL |     20 |
|  7876 | ADAMS | CLERK   | 7788 | 1987-07-13 | 1100 | NULL |     20 |
|  7902 | FORD  | ANALYST | 7566 | 1981-12-03 | 3000 | NULL |     20 |
+-------+-------+---------+------+------------+------+------+--------+
5 rows in set (0.00 sec)
```

But we want how many are working in deptno 20, then it is necessary to go for group functions. Follwing are the group functions.



Aggregate Functions



ename	sal
SMITH	800
JAMES	950
ADAMS	1100
WARD	1250
MARTIN	1250
MILLER	1300
TURNER	1500
ALLEN	1600
CLARK	2450
BLAKE	2850
JONES	2975
FORD	3000
SCOTT	3000
KING	5000
	29025

COUNT

To count the number of rows in an entire table or that match particular conditions, use the COUNT function.
In emp table let us find how many employees are there?

mysql> select count(*),count(empno) from emp;

```
+----------+--------------+
| count(*) | count(empno) |
+----------+--------------+
|       14 |           14 |
+----------+--------------+
```

1 row in set (0.04 sec)

In the above query what is count(*), what is count(empno).? Count(*) includes nulls also where as count(empno) counts number of non-NULL values. (To be frank, empno is a primary key column and hence it cant include nulls definitely). To explain in depth consider the following example.

mysql> select count(*),count(comm) from emp;

```
+----------+-------------+
| count(*) | count(comm) |
+----------+-------------+
|       14 |           4 |
+----------+-------------+
1 row in set (0.00 sec)
```

In the above query we can see only 4 employees taking commission, it is not including null values.

```
If we want to find how many are taking salaries more than 2975
```

```
mysql> select count(empno),count(ename),count(sal),count(hiredate) from emp;
+--------------+--------------+------------+-----------------+
| count(empno) | count(ename) | count(sal) | count(hiredate) |
+--------------+--------------+------------+-----------------+
|           14 |           14 |         14 |              14 |
+--------------+--------------+------------+-----------------+
1 row in set (0.00 sec)

mysql> select count(empno) from emp where sal>2975;
+--------------+
| count(empno) |
+--------------+
|            3 |
+--------------+
1 row in set (0.00 sec)
```

MIN(),MAX()

Finding smallest or largest values in a dataset.

```
mysql> select min(sal) "leastsal",max(sal) "highsal",
       ->min(hiredate) "earliest",max(hiredate) "latest"
       -> from emp;
+----------+---------+------------+------------+
| leastsal | highsal | earliest   | latest     |
+----------+---------+------------+------------+
|      800 |    5000 | 1980-12-17 | 1987-07-13 |
+----------+---------+------------+------------+
1 row in set (0.00 sec)
```

Let us club set option to define the job and rewrite the above query.

```
mysql> set @vjob="salesman";
Query OK, 0 rows affected (0.00 sec)

mysql> select min(sal) "leastsal",max(sal) "highsal",
        ->min(hiredate) "earliest",max(hiredate) "latest"
        -> from emp
        -> where job=@vjob;
+----------+---------+------------+------------+
| leastsal | highsal | earliest   | latest     |
+----------+---------+------------+------------+
|     1250 |    1600 | 1981-02-20 | 1981-09-28 |
+----------+---------+------------+------------+
1 row in set (0.00 sec)
```
Like count min,max also support all datatypes.

mysql> select min(ename),max(ename) from emp;
```
+------------+------------+
| min(ename) | max(ename) |
+------------+------------+
| ADAMS      | WARD       |
+------------+------------+
1 row in set (0.00 sec)
```

SUM(),AVG()

SUM() and AVG() produce the total and average (mean) of a set of values: We want to know what is total salary taken by all the employees or what is the average salary of all employees?

mysql> select sum(sal) "totsal",avg(sal) "averagesal"
-> from emp;
```
+--------+-------------------+
| totsal | averagesal        |
+--------+-------------------+
|  29025 | 2073.214285714286 |
+--------+-------------------+
1 row in set (0.00 sec)
```

Both sum,avg functions work only on numeric datatypes.

GROUP BY

Use a GROUP BY clause to arrange rows into groups.Following statement determines the number of records in emp table, and thus the total number of employees working:
select count(*),count(empno) from emp;
```
+----------+--------------+
| count(*) | count(empno) |
+----------+--------------+
|       14 |           14 |
+----------+--------------+
```

To arrange a set of rows into subgroups and summarize each group, use aggregate functions in conjunction with a GROUP BY clause.

KINDLY USE THIS BEFORE USING GROUP BY

==
set @@sql_mode='ONLY_FULL_GROUP_BY';

==

```
mysql> set @@sql_mode='ONLY_FULL_GROUP_BY';
Query OK, 0 rows affected (0.00 sec)
```

To determine number of employees for each deptno group the rows by deptno.

```
mysql> select count(empno),deptno from emp group by deptno;
+--------------+--------+
| count(empno) | deptno |
+--------------+--------+
|            3 |     10 |
|            5 |     20 |
|            6 |     30 |
+--------------+--------+
3 rows in set (0.00 sec)
```

To determine minimum,maximum salary for each job,group the rows by job.

```
mysql> select min(sal),max(sal),job
    -> from emp
    -> group by job;
+----------+----------+-----------+
| min(sal) | max(sal) | job       |
+----------+----------+-----------+
|      800 |     1300 | CLERK     |
|     1250 |     1600 | SALESMAN  |
|     2450 |     2975 | MANAGER   |
|     3000 |     3000 | ANALYST   |
|     5000 |     5000 | PRESIDENT |
+----------+----------+-----------+
5 rows in set (0.00 sec)
```

To determine total,average salary for each deptno,group the rows by deptno.

```
mysql> select sum(sal),avg(sal),deptno
    -> from emp
    -> group by deptno;
+----------+--------------------+--------+
| sum(sal) | avg(sal)           | deptno |
+----------+--------------------+--------+
|     8750 | 2916.6666666666665 |     10 |
|    10875 |               2175 |     20 |
|     9400 | 1566.6666666666667 |     30 |
+----------+--------------------+--------+
3 rows in set (0.00 sec)
```

GROUPING ON 2 COLUMNS

```
mysql> SELECT sum(sal),deptno,job
    -> from   emp
    -> group by deptno,job
    -> order by 2;
+----------+--------+-----------+
| sum(sal) | deptno | job       |
+----------+--------+-----------+
|     1300 |     10 | CLERK     |
|     2450 |     10 | MANAGER   |
|     5000 |     10 | PRESIDENT |
|     6000 |     20 | ANALYST   |
|     1900 |     20 | CLERK     |
|     2975 |     20 | MANAGER   |
|      950 |     30 | CLERK     |
|     2850 |     30 | MANAGER   |
|     5600 |     30 | SALESMAN  |
+----------+--------+-----------+
9 rows in set (0.00 sec)
```

Selecting Only Groups with Certain Characteristics

To calculate group summaries but display results only for groups that match certain criteria make use of HAVING clause.

I want to find how many have joined in each year first.

```
mysql> select extract(year from hiredate)"year",count(empno)
    -> from emp
    -> group by  extract(year from hiredate);
+------+--------------+
| year | count(empno) |
+------+--------------+
| 1980 |            1 |
| 1981 |           10 |
| 1987 |            2 |
| 1982 |            1 |
+------+--------------+
4 rows in set (0.00 sec)
```

In the above output I want to find count more than 1. HAVING operates on the already-selected-and-grouped set of rows, applying additional constraints based on aggregate function results that aren’t known during the initial selection process.

```
mysql> select extract(year from hiredate),count(empno)
    -> from emp
    -> group by  extract(year from hiredate)
    -> having count(empno)>1;
+-----------------------------+--------------+
| extract(year from hiredate) | count(empno) |
+-----------------------------+--------------+
|                        1981 |           10 |
|                        1987 |            2 |
+-----------------------------+--------------+
2 rows in set (0.00 sec)
```

HIERARCHY OF WRITING QUERIES

SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY

```
 mysql> Select count(empno), extract(month from hiredate) "month"
    -> From emp
    -> Where extract(month from hiredate) not in (4,5)
    -> Group by extract(month from hiredate)
    -> Having count(empno)>1
    -> Order by 2;
+--------------+-------+
| count(empno) | month |
+--------------+-------+
|            2 |     2 |
|            2 |     7 |
|            2 |     9 |
|            3 |    12 |
+--------------+-------+
4 rows in set (0.00 sec)
```

SET OPERATORS

Unlike other RDBMSs Mysql supports only 2 types of set operators.

TYPES

- *UNION*
- *UNION ALL*

Question to be answered is Why we need Set Operators?

Nodoubt select is available to retreive data from tables. Several options are avialable to achieve this.
Consider following example.

```
mysql> select distinct(job) from emp where deptno in(10,20);
+-----------+
| job       |
+-----------+
| MANAGER   |
| PRESIDENT |
| CLERK     |
| ANALYST   |
+-----------+
4 rows in set (0.00 sec)
```

Here we can see first we are finding job available in deptno 10,20 and applying distinct operator. This also can be accomplished using set operator.

```
mysql> select job from emp where deptno=10
    -> union
    -> select job from emp where deptno=20;
+-----------+
| job       |
+-----------+
| MANAGER   |
| PRESIDENT |
| CLERK     |
| ANALYST   |
+-----------+
4 rows in set (0.00 sec)
```

Current query data is also sorted.

Set operators combines the result from multiple SELECT statements into a single result set. The result set column names are taken from the column names of the first SELECT statement.

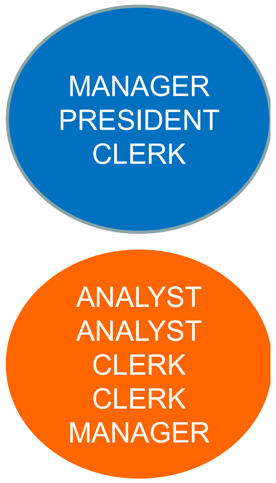
The SET Operators

UNION

SELECT JOB FROM EMP WHERE DEPTNO=10

UNION

SELECT JOB FROM EMP WHERE DEPTNO=20



JOB
ANALYST
CLERK
MANAGER
PRESIDENT

Rules for using set operators

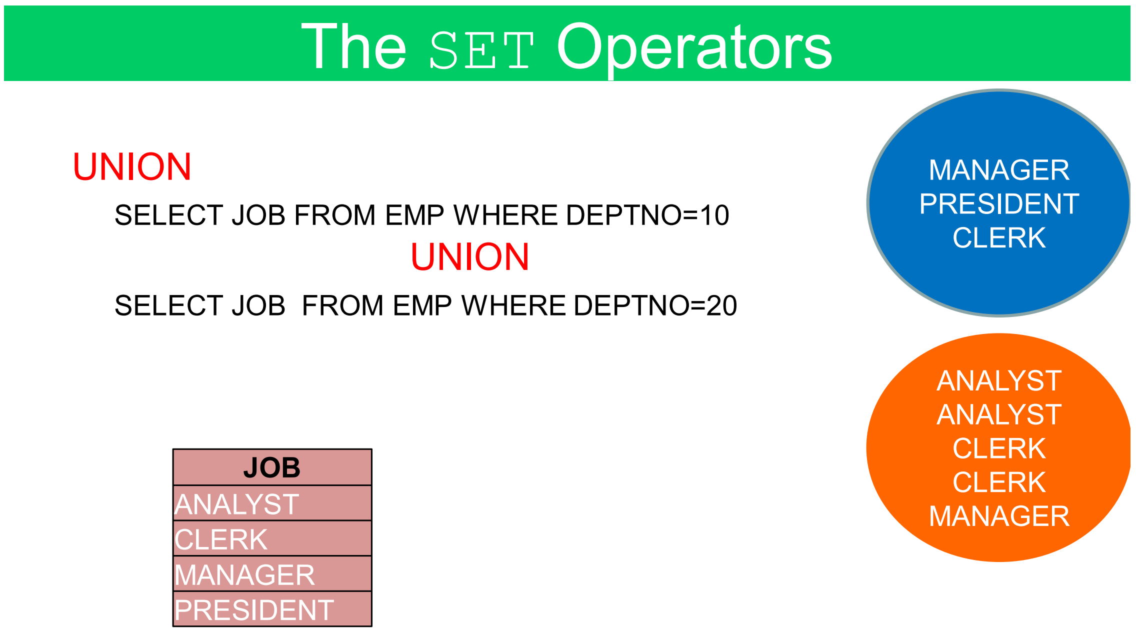
Number of columns in the select statements should be same.

```
mysql> select job,deptno from emp where deptno=10
    -> union
    -> select job from emp where deptno=20;
```

ERROR 1222 (21000): The used SELECT statements have a different number of columns

```
mysql> select job from emp where deptno=10
```

```
    -> union all
    -> select job from emp where deptno=20;
+------------+
| job        |
+------------+
| MANAGER    |
| PRESIDENT  |
| CLERK      |
| CLERK      |
| MANAGER    |
| ANALYST    |
| CLERK      |
| ANALYST    |
+------------+
8 rows in set (0.00 sec)
```



The SET Operators

UNION

SELECT JOB FROM EMP WHERE DEPTNO=10

UNION

SELECT JOB FROM EMP WHERE DEPTNO=20

JOB
ANALYST
CLERK
MANAGER
PRESIDENT

Look at the differences between union and and union all

UNION	UNION ALL
NO DUPLICATES	ALLOWS DUPLICATES
SORTS THE DATA	NO SORTING

QUALIFYING NUMBER OF COLUMNS USING NULL

Consider following query.

```
mysql> select ename from emp
    -> union
    -> select dname from dept;
+------------+
| ename      |
+------------+
| SMITH      |
| ALLEN      |
| WARD       |
| JONES      |
| MARTIN     |
| BLAKE      |
| CLARK      |
| SCOTT      |
| KING       |
| TURNER     |
| ADAMS      |
| JAMES      |
| FORD       |
| MILLER     |
| ACCOUNTING |
| RESEARCH   |
| SALES      |
| OPERATIONS |
+------------+
18 rows in set (0.00 sec)
```

IN SET OPERATORS HEADING ALWAYS COMES FROM 1ST SELECT STATEMENT

As we can see output looks clumsy. We are unable to make out properly.

```
mysql> select ename,null dname from emp
    -> union
    -> select null,dname from dept;
+--------+------------+
| ename  | dname      |
+--------+------------+
| SMITH  | NULL       |
| ALLEN  | NULL       |
| WARD   | NULL       |
| JONES  | NULL       |
| MARTIN | NULL       |
| BLAKE  | NULL       |
| CLARK  | NULL       |
| SCOTT  | NULL       |
| KING   | NULL       |
| TURNER | NULL       |
| ADAMS  | NULL       |
| JAMES  | NULL       |
| FORD   | NULL       |
| MILLER | NULL       |
| NULL   | ACCOUNTING |
| NULL   | RESEARCH   |
| NULL   | SALES      |
| NULL   | OPERATIONS |
+--------+------------+
18 rows in set (0.00 sec)
```

ORDER BY WITH SET OPERATOR

```
mysql> select ename,null dname from emp
    -> union
    -> select null,dname from dept
    -> order by 1;
+--------+------------+
| ename  | dname      |
+--------+------------+
| NULL   | ACCOUNTING |
| NULL   | RESEARCH   |
| NULL   | SALES      |
| NULL   | OPERATIONS |
| ADAMS  | NULL       |
| ALLEN  | NULL       |
| BLAKE  | NULL       |
| CLARK  | NULL       |
| FORD   | NULL       |
| JAMES  | NULL       |
| JONES  | NULL       |
| KING   | NULL       |
| MARTIN | NULL       |
| MILLER | NULL       |
| SCOTT  | NULL       |
| SMITH  | NULL       |
| TURNER | NULL       |
| WARD   | NULL       |
+--------+------------+
```

18 rows in set (0.00 sec)

Order by clause should be put after all select statements yet it works only for 1st select statement.

JOINS

Except set operators remaining queries that is worked so far is with respect to single table. Applications require data from multiple tables.

• To hold intermediate results for a multiple-stage operation
• To modify rows in one table based on information from another
• To combine rows from tables to obtain more comprehensive information than that can be obtained from individual tables alone.

Joins can be used to find matched data or mismatched data. Since oracle was the first RDBMS, join syntaxes can be ORACLE PROPRIETARY syntax or SQL-99 Syntax. Here our endevour is towards SQL-99 syntax.

Tables used for explaining different type of joins.

EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500		30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7682	23-JAN-82	1300		10

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEWYORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

SALGRADE

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

JOIN TYPES

INNER

OUTER

NON EQUI JOIN

CROSS

SELF

NATURAL

USING

J O I N T Y P E S

RULES

COLUMN NAMES SHOULD HAVE THE FORMAT

E.ENAME *EMP.ENAME*

TYPE OF JOIN TO BE SPECIFIED

JOIN CONDITION USES "ON" CLAUSE

INNER JOIN

Produce only the relevant matches by including appropriate join conditions. It makes use of '=' operator for joining condition.

Suppose I want to find dname in which Mr.Allen is working, dname is available in dept table Allen is available in emp table. Between both tables we have deptno in common.

JOIN TYPES

INNER

Uses = operator for joining condition

EMP

KING	10
JONES	20
BLAKE	30

=

DEPT

DEPTNO	DNAME
10	ACCOUNTING
20	RESEARCH
30	SALES
40	OPERATIONS

```
SELECT       E.ENAME,D.DNAME
FROM         EMP E
INNER JOIN   DEPT D
ON           E.DEPTNO=D.DEPTNO
```

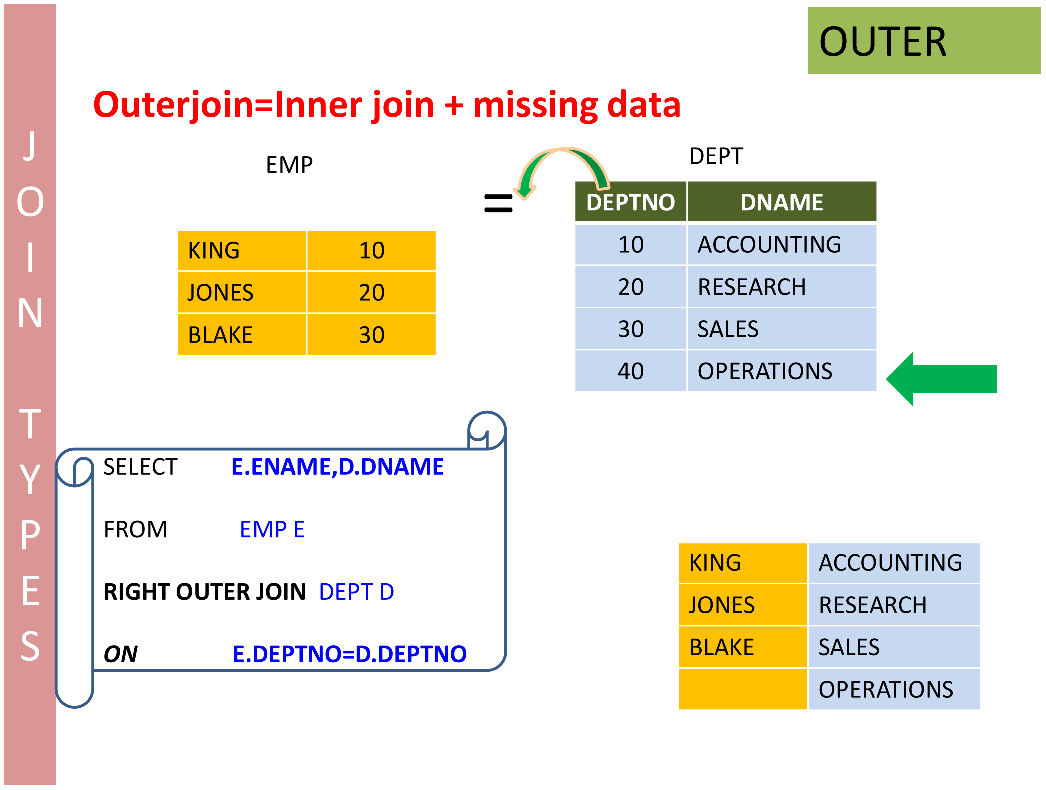
KING	ACCOUNTING
JONES	RESEARCH
BLAKE	SALES

Additional conditions can be implemented by making use of WHERE/AND.

When we perform inner join matched rows only will come. Any missing data also to be obtained, it is required to go for OUTER JOIN.

OUTER JOIN

To produce a list on the basis of a join between tables, and you want the list to include an entry for every row in the first table, including those for which no match occurs in the second table.



J O I N T Y P E S

OUTER

Outerjoin=Inner join + missing data

EMP

KING	10
JONES	20
BLAKE	30

DEPT

DEPTNO	DNAME
10	ACCOUNTING
20	RESEARCH
30	SALES
40	OPERATIONS

```
SELECT          E.ENAME,D.DNAME
FROM            EMP E
RIGHT OUTER JOIN  DEPT D
ON              E.DEPTNO=D.DEPTNO
```

KING	ACCOUNTING
JONES	RESEARCH
BLAKE	SALES
	OPERATIONS

Outer join can be left or right(depending upon which table is having missing data). In the above slide we can see that OPERATIONS row is the missing data which is available in dept table. Apart from emp table data from dept we wanted entire set of rows from dept table.

Below slide shows missing data available in left hand side(emp) table.

J O I N

T Y P E S

OUTER

Outerjoin=Inner join + missing data

EMP

KING	10
JONES	20
BLAKE	30
RAM	

=

DEPT

DEPTNO	DNAME
10	ACCOUNTING
20	RESEARCH
30	SALES
40	OPERATIONS

```
SELECT          E.ENAME,D.DNAME

FROM            EMP E

LEFT OUTER JOIN  DEPT D

ON              E.DEPTNO=D.DEPTNO
```

KING	ACCOUNTING
JONES	RESEARCH
BLAKE	SALES
RAM	

FULL Outer join syntax not available in Mysql. Of course it can be obtained using combination of left and right outer join with UNION operator.

JOIN TYPES

OUTER

Outerjoin=Inner join + missing data

EMP

KING	10
JONES	20
BLAKE	30
RAM	

DEPT

DEPTNO	DNAME
10	ACCOUNTING
20	RESEARCH
30	SALES
40	OPERATIONS

```
SELECT          E.ENAME,D.DNAME
FROM            EMP E
LEFT OUTER JOIN DEPT D
ON              E.DEPTNO=D.DEPTNO
UNION
SELECT          E.ENAME,D.DNAME
FROM            EMP E
RIGHT OUTER JOIN DEPT D
ON              E.DEPTNO=D.DEPTNO
```

KING	ACCOUNTING
JONES	RESEARCH
BLAKE	SALES
	OPERATIONS
RAM	

CROSS JOIN

A complete join that produces all possible row combinations is called cross join.

J O I N T Y P E S

EMP

KING	10
JONES	20
BLAKE	30

DEPT

DEPTNO	DNAME
10	ACCOUNTING
20	RESEARCH
30	SALES
40	OPERATIONS

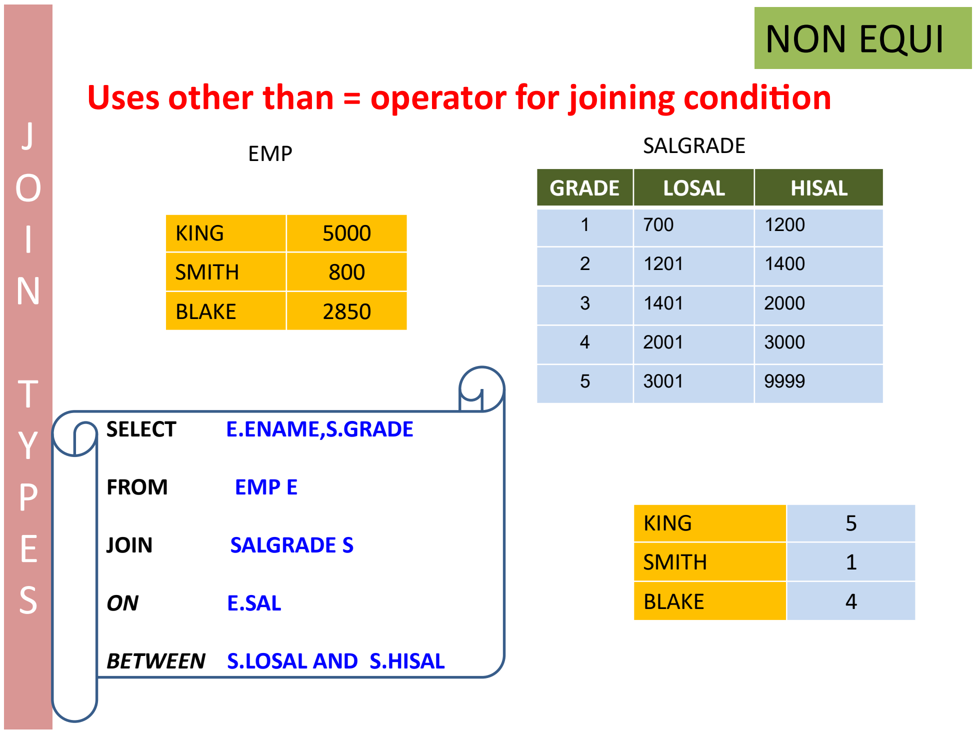
CROSS

```
SELECT      E.ENAME,D.DNAME
FROM         EMP E
CROSS JOIN  DEPT D
```

KING	ACCOUNTING
JONES	ACCOUNTING
BLAKE	ACCOUNTING
KING	RESEARCH
JONES	RESEARCH
BLAKE	RESEARCH
KING	SALES
JONES	SALES
BLAKE	SALES
KING	OPERATIONS
JONES	OPERATIONS
BLAKE	OPERATIONS

NON EQUI JOIN

Join uses other than '=' operator.



JOIN TYPES

NON EQUI

Uses other than = operator for joining condition

EMP

KING	5000
SMITH	800
BLAKE	2850

SALGRADE

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

```
SELECT    E.ENAME,S.GRADE
FROM      EMP E
JOIN      SALGRADE S
ON        E.SAL
BETWEEN   S.LOSAL AND  S.HISAL
```

KING	5
SMITH	1
BLAKE	4

In the above example as you can see between operator is used for join condition.

SELF JOIN

Sometimes the table itself has to be joined for getting desired results. Take for example SMITH is reporting to 7902. I want to know name of 7902 which is available in the same table unlike previous cases where in we were searching in different tables.

J O I N   T Y P E S

SELF

Table is joined to the table itself

EMP

EMPNO	ENAME	MGR
7369	SMITH	7902
7902	FORD	7566
7566	JONES	7839
7839	KING	-

=

EMP

EMPNO	ENAME	MGR
7369	SMITH	7902
7902	FORD	7566
7566	JONES	7839
7839	KING	-

```
SELECT    E.ENAME "empname",
          M.ENAME  "manager"
FROM       EMP E

JOIN       EMP M

ON         E.MGR=M.EMPNO
```

SMITH	FORD
FORD	JONES
JONES	KING

NATURAL JOIN

Tables are joined naturally on common columns. It works like a inner join barring join condition cannot be specified

NATURAL

*Naturally tables joined on common columns

- Common columns in both the tables should have same datatype
- Cant specify column on which join should be done

```
SELECT          ENAME,DNAME

FROM             EMP

NATURAL JOIN    DEPT
```

USING

USING

* **When multiple columns are there we can specify which column should be used for joining**

• **Suitable for Common columns in both the tables with different datatype**

```
SELECT      ENAME,DNAME
FROM        EMP
JOIN        DEPT
USING       (DEPTNO)
```

SUBQUERIES

Having learnt aggregate functions, we are conversant about max function. Suppose we want to find maximum salary with respect to emp table. Query to accomplish this is

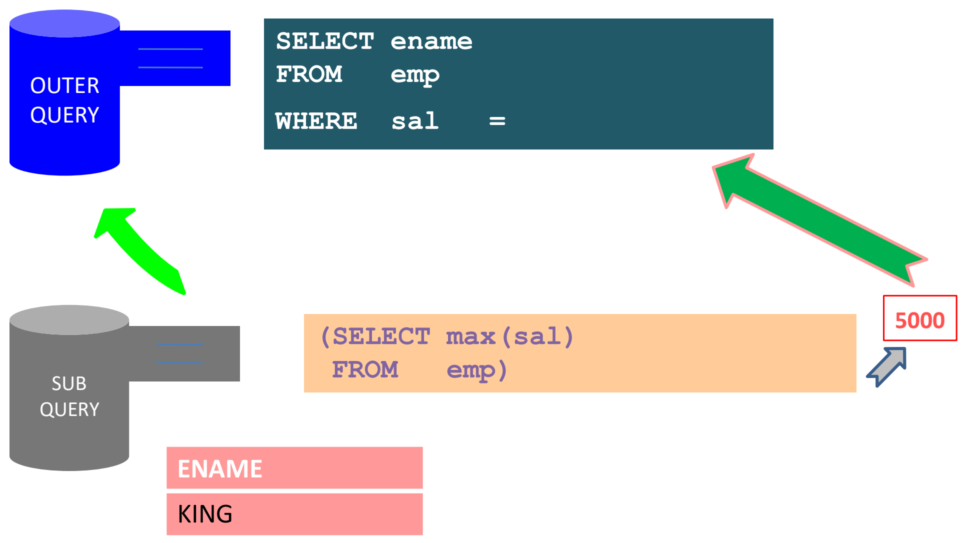
```
mysql> select max(sal) from emp;
+----------+
| max(sal) |
+----------+
|     5000 |
+----------+
1 row in set (0.02 sec)
```

But, if it is required to find the name of the employee getting this maximum salary, how to go about this?
We need One More Select statement definitely. Earlier query shown should be put inside this select statement (In ohter words SELECT INSIDE SELECT)

RULES

1. Subquery should be enclosed in brackets
2. Subquery executes 1st and gives the result
3. Outer query uses this result and gets executed

Subquery



```
SELECT ename
FROM   emp
WHERE  sal   =
```

```
(SELECT max(sal)
 FROM   emp)
```

ENAME
KING

Subquery can be of 5 types. We will go in depth of each.

Subquery

TYPES

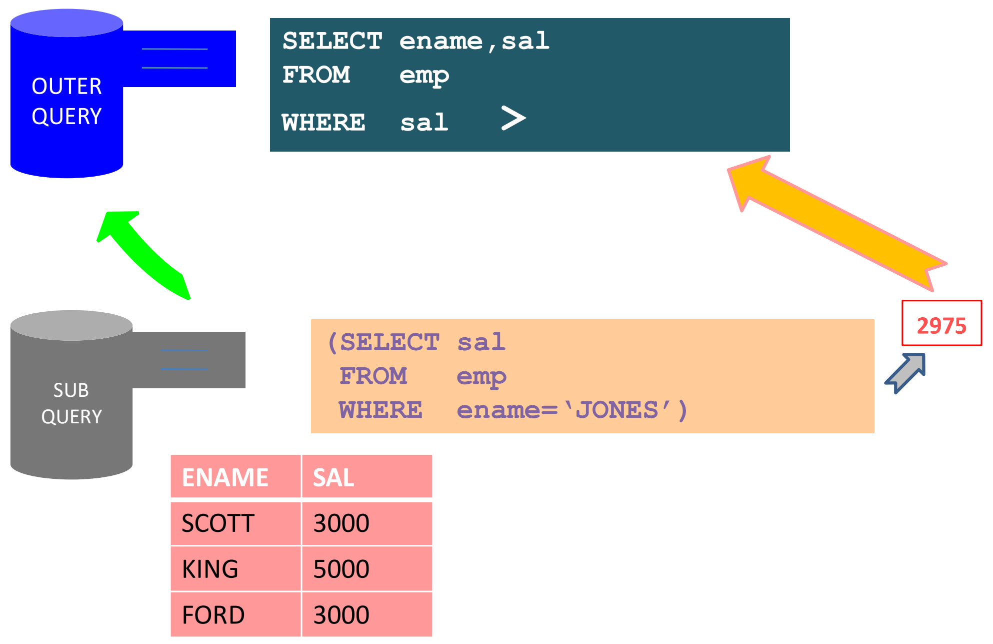
- *SINGLE ROW*
- *MULTIPLE ROW*
- *MULTIPLE COLUMN*
- *NESTED*
- *CORRELATED*

SINGLE ROW SUBQUERY

Let us find out the employees details who are taking more than Mr. Jones salary. As per emp table current data, only one employee with Jones name is available. Since subquery executes 1st, we need to find Jones salary which gives only1 row.

SUBQUERY RETURNS SINGLE ROW

Subquery



```
SELECT ename,sal
FROM   emp
WHERE  sal   >
```

```
(SELECT sal
 FROM   emp
 WHERE  ename='JONES')
```

ENAME	SAL
SCOTT	3000
KING	5000
FORD	3000

MULTIPLE ROW SUBQUERY

Having seen single row query which returns single row, assume that there is a task to find employees who are taking same salaries as FORD,WARD. Obviously subquery has to find SALARY of FORD,WARD returning *More than ONE row.* If we use = operator in the outer query it will return an error.

SUBQUERY RETURNS MORE THAN ONE ROW

```
mysql> select ename,sal
    -> from   emp
    -> where  sal=(select sal from emp
    ->             where ename in('WARD','MARTIN'));
```

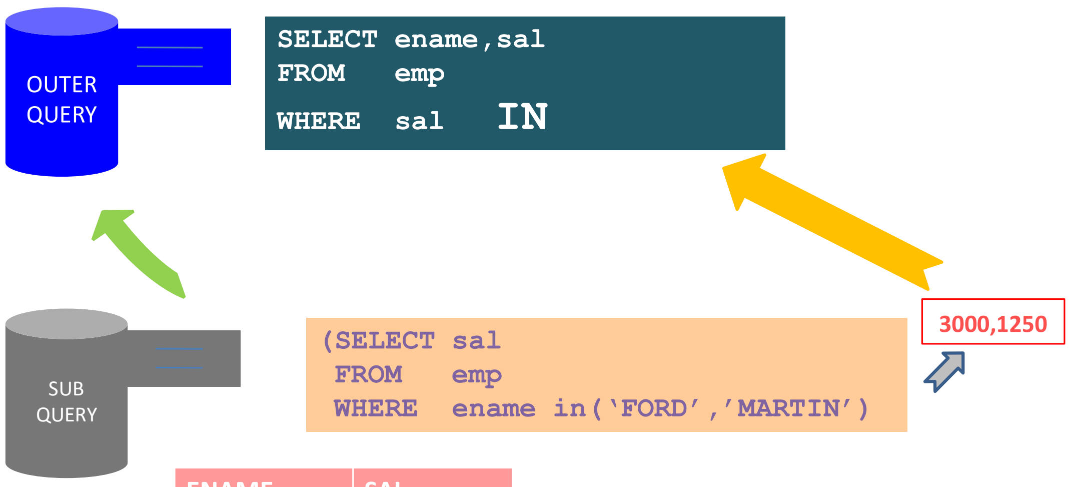
ERROR 1242 (21000): Subquery returns more than 1 row

The error is because = takes single value. If at all it has to take multiple values we need to ***change operator.*** *Operators recommended wrt single row and multiple row subquery are as below*

SUBQUERY	OPERATORS
SINGLE ROW	=,<,>,<=,>=,<>
MULTIPLE ROW	IN,NOT IN,ANY,ALL

Subquery

Multiple row subquery using IN



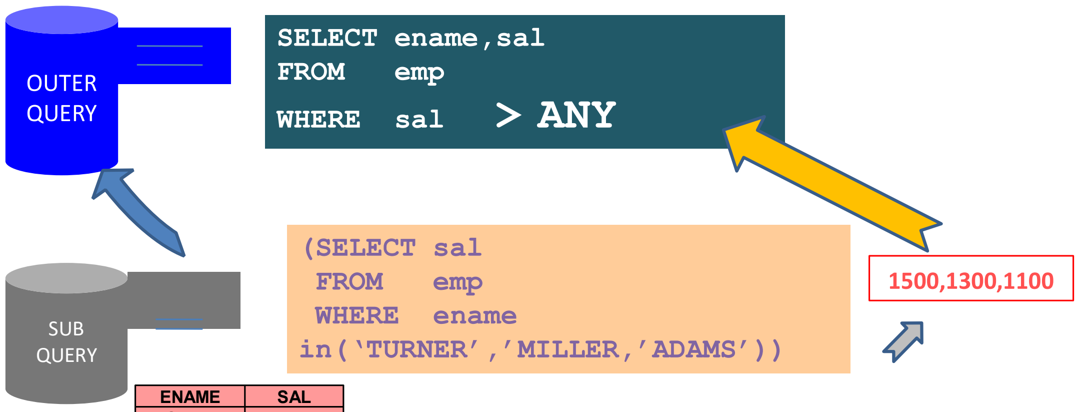
```
SELECT ename,sal
FROM   emp
WHERE  sal    IN
```

```
(SELECT sal
 FROM   emp
 WHERE  ename in('FORD','MARTIN')
```

ENAME	SAL
SCOTT	3000
WARD	1250

Subquery

Multiple row subquery using ANY



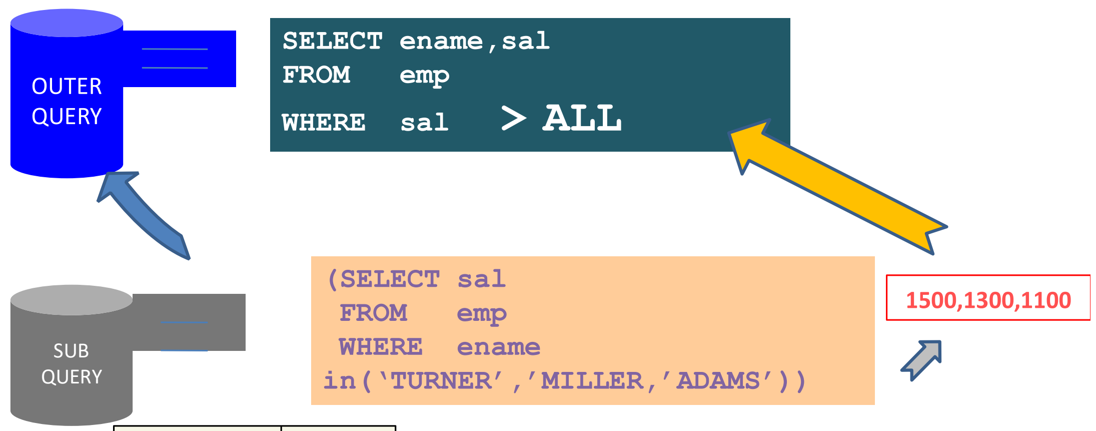
```
SELECT ename,sal
FROM   emp
WHERE  sal   > ANY
```

```
(SELECT sal
 FROM   emp
 WHERE  ename
in(‘TURNER’,’MILLER,’ADAMS’))
```

ENAME	SAL
KING	5000
FORD	3000
SCOTT	3000
JONES	2975
BLAKE	2850
CLARK	2450
ALLEN	1600
TURNER	1500
MILLER	1300
WARD	1250
MARTIN	1250

Subquery

Multiple row subquery using ALL



```
SELECT ename,sal
FROM   emp
WHERE  sal    > ALL
```

```
(SELECT sal
 FROM   emp
 WHERE  ename
in('TURNER','MILLER,'ADAMS'))
```

ENAME	SAL
ALLEN	1600
CLARK	2450
BLAKE	2850
JONES	2975
SCOTT	3000
FORD	3000
KING	5000

MULTIPLE COLUMN SUBQUERY

Subqueries so far seen were returning single row/multiple row but ***ONLY ONE COLUMN.***

Here is a subquery which gives 2 columns.

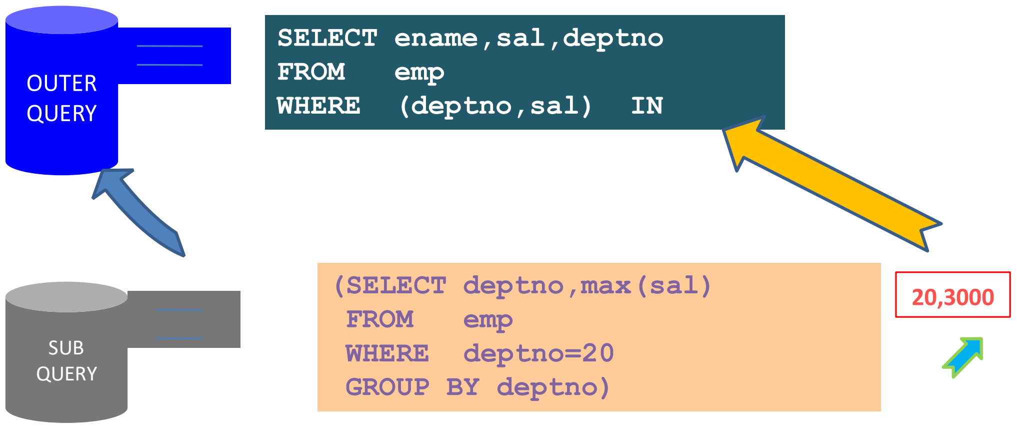
```
mysql> select max(sal),deptno
    -> from   emp
    -> group by deptno;
+----------+--------+
| max(sal) | deptno |
+----------+--------+
|     5000 |     10 |
|     3000 |     20 |
|     2850 |     30 |
+----------+--------+
3 rows in set (0.01 sec)
```

SUBQUERY RETURNS MORE THAN ONE COLUMN

We wanted to find employees who are taking maximum salary in each deptno;

Subquery

Multiple COLUMN subquery



```
SELECT ename,sal,deptno
FROM   emp
WHERE  (deptno,sal)  IN
```

```
(SELECT deptno,max(sal)
 FROM   emp
 WHERE  deptno=20
 GROUP BY deptno)
```

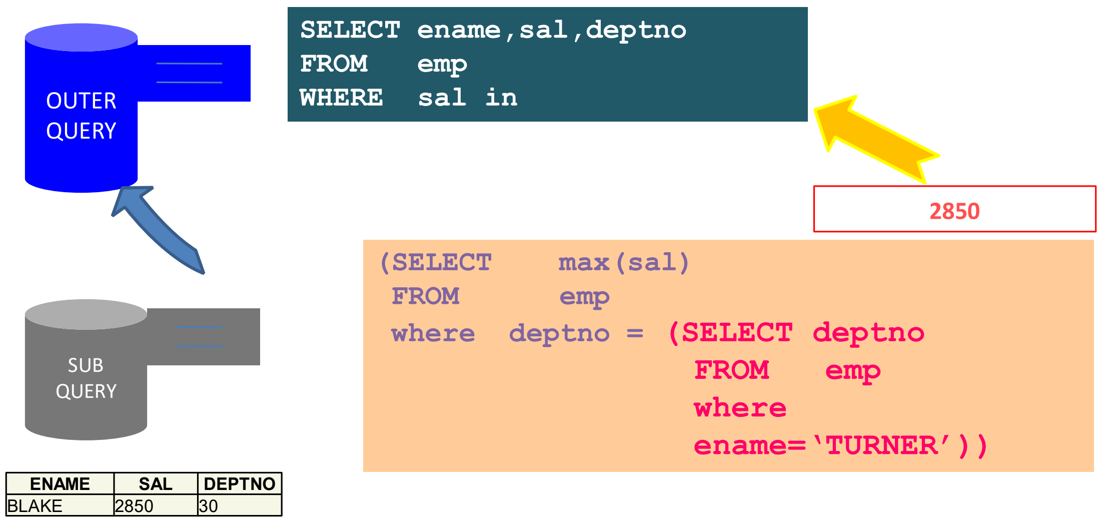
ENAME	SAL	DEPTNO
FORD	3000	20
SCOTT	3000	20

NESTED SUBQUERY

Subquery can return single row,multiple row with single column or multiple columns. It may be required that subquery itself needs another subquery which we refer as nested query.

Subquery

NESTED subquery



```
SELECT ename,sal,deptno
FROM   emp
WHERE  sal in
```

```
(SELECT    max(sal)
 FROM       emp
 where   deptno =  (SELECT deptno
                    FROM    emp
                    where
                    ename='TURNER'))
```

ENAME	SAL	DEPTNO
BLAKE	2850	30

CORRELATED SUBQUERY

The correlation comes from the fact that the subquery uses information from the outer query and the subquery executes once for every row in the outer query. We can easily write employees who are taking salaries more than the average salary of entire table.

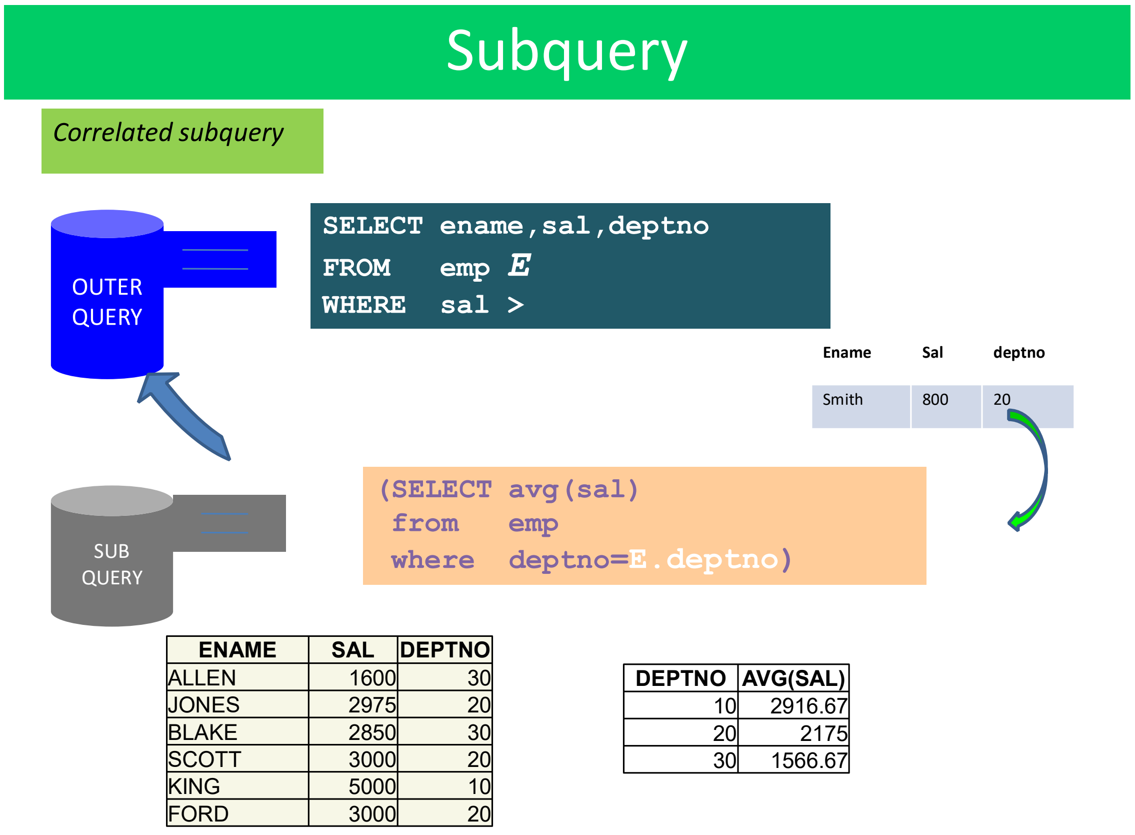
IF AT ALL IT IS REQUIRED TO FIND EMPLOYEES WHO ARE TAKING SALARIES MORE THAN THE RESPECTIVE DEPTNO AVERAGE SALARY.

```
mysql> SELECT ENAME,SAL,DEPTNO FROM EMP
    -> WHERE DEPTNO=10;
+--------+------+--------+
| ENAME  | SAL  | DEPTNO |
+--------+------+--------+
| CLARK  | 2450 |     10 |
| KING   | 5000 |     10 |
| MILLER | 1300 |     10 |
+--------+------+--------+
```

```
mysql> SELECT AVG(SAL),DEPTNO FROM EMP GROUP BY DEPTNO;
+--------------------+--------+
| AVG(SAL)           | DEPTNO |
+--------------------+--------+
| 2916.6666666666665 |     10 |
|               2175 |     20 |
| 1566.6666666666667 |     30 |
+--------------------+---------+
```

To explain we have taken only deptno 10 details. As it is seen that there are 3 employees in deptno 10. Average salary for deptno 10 is 2916.66... As per the requirement EMPLOYEE *WHO IS WORKING IN DEPTNO 10 SHOULD BE TAKING MORE THAN DEPTNO 10 AVERAGE SALARY.*

Like wise who is working in 20 should be taking more than deptno20 average salary and deptno 30 people should be drawing more than deptno 30 average salary.



Subquery

Correlated subquery

```
SELECT ename,sal,deptno
FROM   emp E
WHERE  sal >
```

Ename	Sal	deptno
Smith	800	20

```
(SELECT avg(sal)
 from   emp
 where  deptno=E.deptno)
```

ENAME	SAL	DEPTNO
ALLEN	1600	30
JONES	2975	20
BLAKE	2850	30
SCOTT	3000	20
KING	5000	10
FORD	3000	20

DEPTNO	AVG(SAL)
10	2916.67
20	2175
30	1566.67

It demonstrates that the subquery uses data from the outer query and the subquery executes once for every row in the outer query.

1. **The outer query passes a value for deptno to the subquery. It takes place in the WHERE clause in the subquery [where deptno = E.deptno]**

2. **The subquery uses this passed-in deptno value to look up the average salary for this deptno [select avg(sal) from emp]**

3. **When the average salary for the deptno is found in the subquery, it's returned to the outer query.**

4. **The outer query then uses this average salary in its WHERE clause to find sal more than it [where sal>]**

When the row is found, query engine temporarily holds the row in memory. It's guaranteed that a row will be found because both outer query and subquery use the same table - emp.

The query engine then moves onto next row in EMP table and repeat Step 1 to 3 again for the next deptno;