UPPSALA UNIVERSITET

SQL – Data Manipulation Language Part 1

lecturer: Mani Pelmo/Neena Thota

mpelmo@sherubtse.edu.bt

Neena.Thota@it.uu.se

UPPSALA UNIVERSITET

Intended Learning Outcomes

- Write simple **SELECT-FROM-WHERE** queries.
- Specify simple JOIN queries.
- Use basic operators (*, is null, between, like, aliases).
- Order the result.
- Write queries involving set operators.
- Apply aggregate functions to groups of tuples.
- Write queries with aggregation
  - GROUP BY / HAVING.

A simple SQL query

```
select <attribute_list>
from <table_name>
where <predicate>
```

1) A1, A2, A3
2) *

1) No WHERE clause: same as WHERE TRUE.
2) Boolean expression (e.g., “A1>A2 and A3<>A1”)

```
select Location, Project
from Location
where Project = 2
```

Eg: Select all locations of project 2.

Other predicates

select *<attribute_list>*
from *<list_of_tables>*
where *<predicate>*

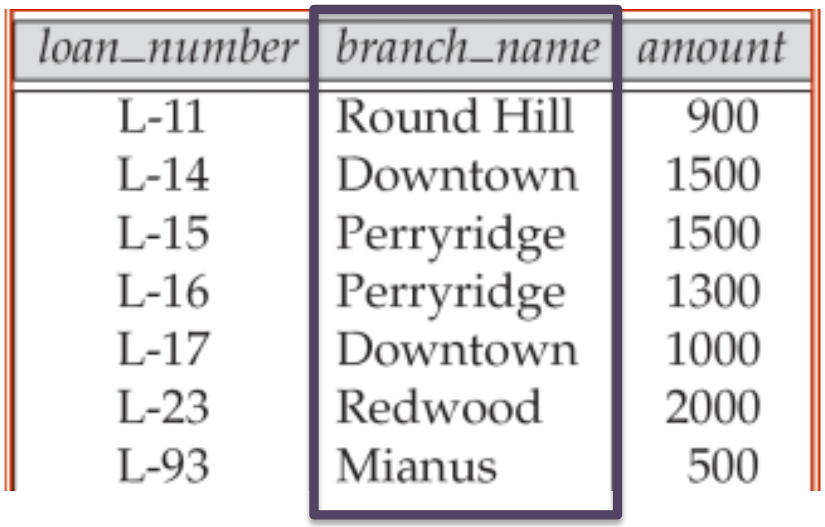
1) A1, A2, A3
2) *

1) No WHERE clause: same as WHERE TRUE
2) Boolean expression (e.g., A1>A2 and A3<>A1)
3) **Boolean expression + math (e.g., A1+A2 < A3*A1)**
4) **A BETWEEN Value1 AND Value2**
5) **A LIKE 'pattern', where pattern can contain _ (any character) and % (any string of characters), e.g., "Ma__e%"**
6) **A IS NULL or A IS NOT NULL**

The SELECT clause

- The **SELECT** clause is used to **list the attributes desired in the result of a query**.
- **Eg:** Find the names of all branches in the *loan* relation:

loan



loan_number	*branch_name*	*amount*
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

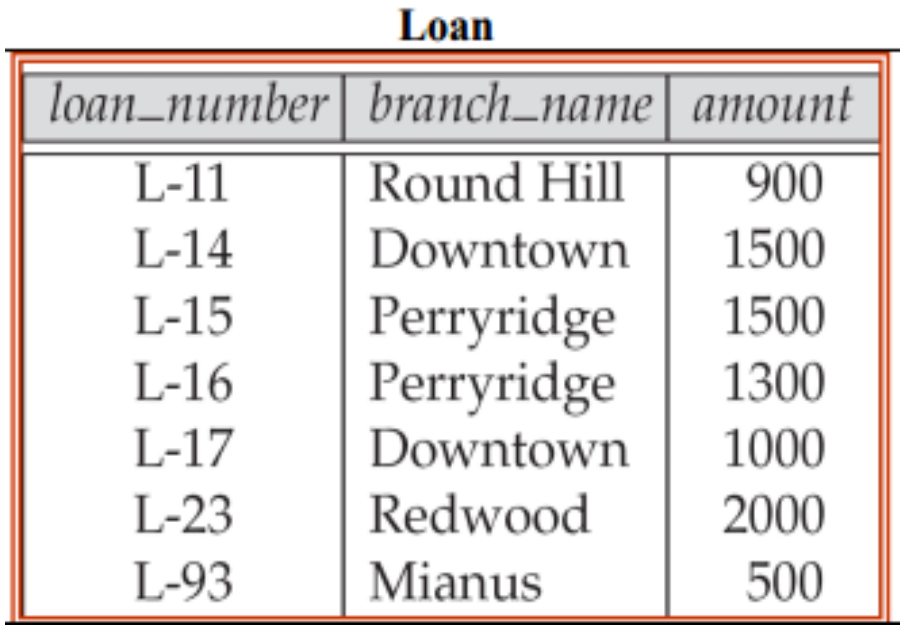
SELECT branch_name **FROM** loan ;

- An asterisk (*) in the select clause denotes "all attributes".

SELECT * **FROM** loan ;

The SELECT clause (cont.)

- SQL allows duplicates in relations as well as in query results. To force the elimination of duplicates, insert the keyword **DISTINCT** after select.
- **Eg:** Find the names of all branches in the loan relation, and remove duplicates:



Loan

loan_number	*branch_name*	*amount*
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

SELECT ***DISTINCT*** branch_name **FROM** loan ;

UPPSALA UNIVERSITET

The SELECT clause (cont.)

- The SELECT clause can also contain arithmetic expressions involving the operators, **+**, **-**, *, and /, operating on constants or attributes of tuples.
- **Eg:** Return the loan relation where the amount attribute multiplied by 100:

Loan

loan_number	branch_name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

```
SELECT branch_name, loan_number, amount * 100
    FROM loan ;
```

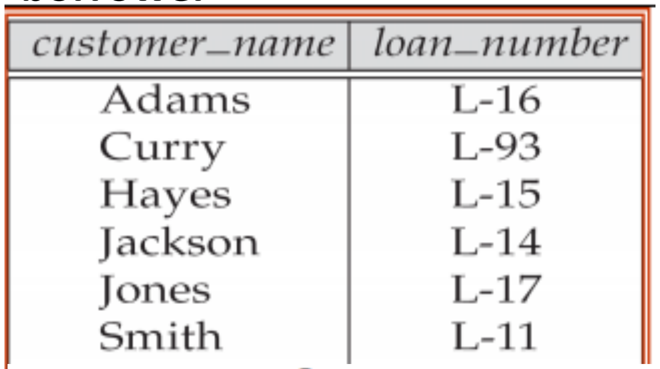
- The keyword **ALL** specifies that duplicates will not be removed:

```
SELECT ALL branch_name FROM loan ;
```

The FROM clause

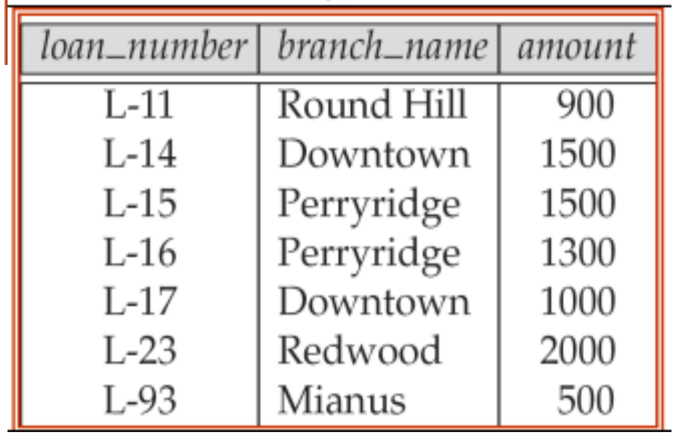
- The **FROM** clause corresponds to the *Cartesian product* operation of the relational algebra. It lists the relations to be scanned when evaluating the whole SELECT expression.
- **Eg:** Find the Cartesian product borrower × loan:

borrower



customer_name	*loan_number*
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11

Loan



loan_number	*branch_name*	*amount*
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

```
SELECT * FROM borrower, loan ;
```

UPPSALA UNIVERSITET

The FROM clause (cont.)

borrower

customer_name	*loan_number*
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

Loan

loan_number	*branch_name*	*amount*
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

- **Eg:** Find the name and loan number of all customers having a loan at the Perryridge branch.

```
SELECT DISTINCT customer_name,
   borrower.loan_number
FROM borrower, loan
WHERE borrower.loan_number = loan.loan_number AND
          branch_name = "Perryridge;
```

Join condition

Selection condition

The WHERE clause

- The **WHERE** clause corresponds to the selection predicate of the relational algebra. It consists of a predicate involving attributes of the relations that appear in the **FROM** clause.

- SQL uses the logical connectives **AND**, **OR**, (and **NOT**). It allows the use of arithmetic expressions as operands to the comparison operators.

UPPSALA
UNIVERSITET

The WHERE clause (cont.)

Eg: Find all loan numbers for loans made at the Perryridge branch with loan amounts greater than $1200:

```
SELECT loan_number  FROM loan
WHERE branch_name = “Perryridge”  AND
amount > 1200 ;
```

The WHERE clause (cont.)

- SQL includes a **BETWEEN** comparison operator in order to simplify **WHERE** clauses that specify that a value is less than or equal to some value and greater than or equal to some other value.

Eg: Find the loan number of those loans with loan amounts between \$90,000 and \$100,000 (that is, ≥ \$90,000 AND ≤ \$100,000)

```
SELECT loan_number  FROM loan
WHERE amount BETWEEN 90000 AND 100000 ;
```

Simple Joins

select <attribute_list>
from <list_of_tables>
where <predicate>

1) A1, A2, A3
2) *

1) T
2) **T1, T2 (computes the cross product of T1 and T2)**
3) **T1 join T2 on <join_condition>**

1) No WHERE clause: same as WHERE TRUE
2) Boolean expression (e.g., A1>A2 and A3<>A1)
3) Boolean expression + math (e.g., A1+A2 < A3*A1)
4) A BETWEEN Value1 AND Value2
5) A LIKE ‘pattern’, where pattern can contain _ (any character) and % (any string of characters)
6) A IS NULL or A IS NOT NULL

Eg: Select all the EmpIDs of Department managers, indicating the corresponding department name.

select EmpID, Name
from Manager, Department
where Department = Number

select EmpID, Name
from Manager **join** Department **on** Department = Number

UPPSALA UNIVERSITET

The RENAME operation

- The SQL mechanism for renaming relations and attributes is accomplished through the **AS** clause:

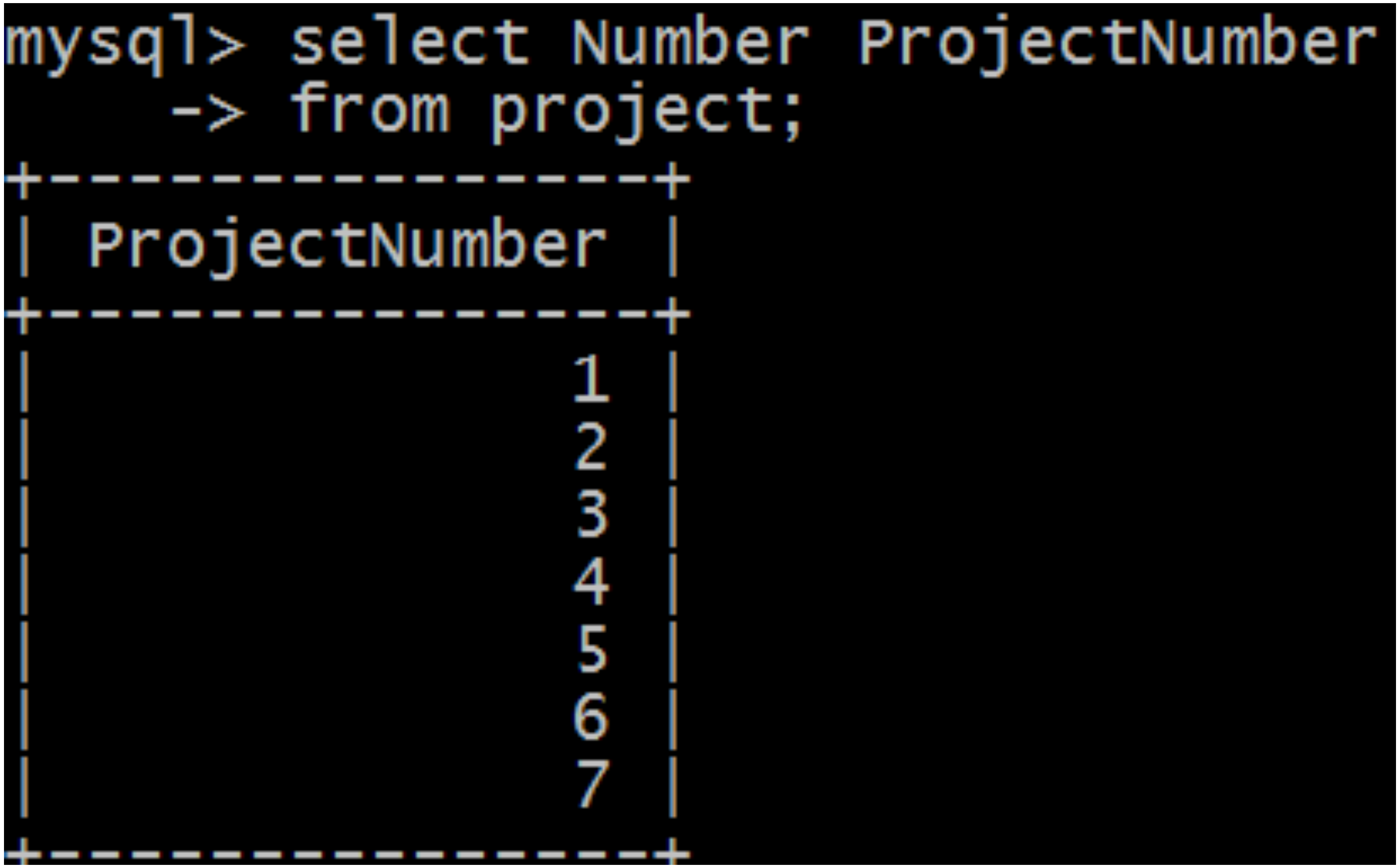
  old-name **AS** *new-name*

- **Eg:** Find the name and loan number of all customers having a loan at the Perryridge branch; replace the column name loan_number with the name lid.

```
SELECT DISTINCT customer_name,
   borrower.loan_number  AS lid
FROM borrower, loan
WHERE borrower.loannumber = loan.loan_number  AND
        branch_name = “Perryridge” ;
```

UPPSALA UNIVERSITET

The RENAME operation:Eg.



```
mysql> select Number ProjectNumber
    -> from project;
+---------------+
| ProjectNumber |
+---------------+
|             1 |
|             2 |
|             3 |
|             4 |
|             5 |
|             6 |
|             7 |
+---------------+
```

UPPSALA UNIVERSITET

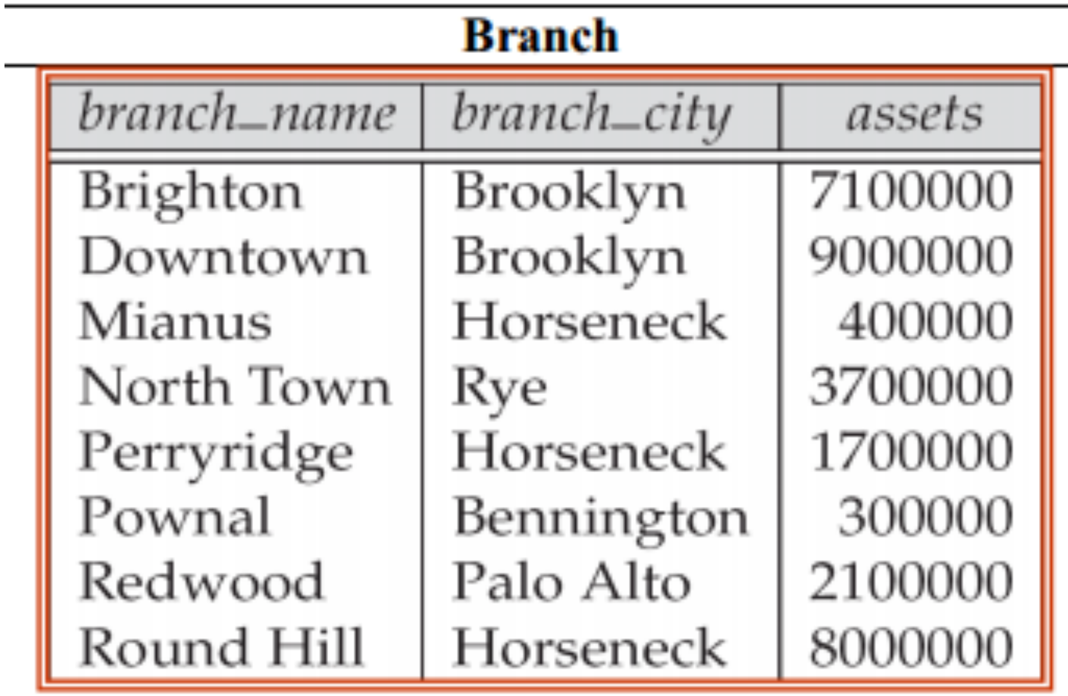
Tuple variables

- **Tuple variables (aliases)** are defined in the FROM clause via the use of the AS clause.
- **Eg:** Find the customer names and their loan numbers for all customers having a loan at some branch.

```
SELECT DISTINCT customer_name, T.loan_number
FROM borrower AS T, loan AS S
WHERE T.loan_number = S.loan_number ;
```

Tuple variables (cont.)

- **Eg:** Find the names of all branches that have greater assets than some branch located in Brooklyn.



Branch

branch_name	*branch_city*	*assets*
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
North Town	Rye	3700000
Perryridge	Horseneck	1700000
Pownal	Bennington	300000
Redwood	Palo Alto	2100000
Round Hill	Horseneck	8000000

```
SELECT DISTINCT T.branch_name
FROM branch AS T, branch AS B
WHERE T.assets > B.assets AND B.branch_city =“Brooklyn”;
```

UPPSALA UNIVERSITET

Aliases / Renaming

```
select EmpID, Name
from Manager, Department
where Department = Number
```

```
select Manager.EmpID, Department.Name
from Manager, Department
where Manager.Department = Department.Number
```

```
select M.EmpID AS Man, D.Name AS DeptName
from Manager M, Department D
where M.Department = D.Number
```

String Operations

- SQL includes a string-matching operator for comparisons on character strings.
- Patterns are described using two special characters:
  - percent (%) . The **%** character matches any substring.
  - underscore (_). The **_** character matches any character.

String Operations (cont.)

- **Eg:** Find the names of all customers whose street includes the substring "Main":

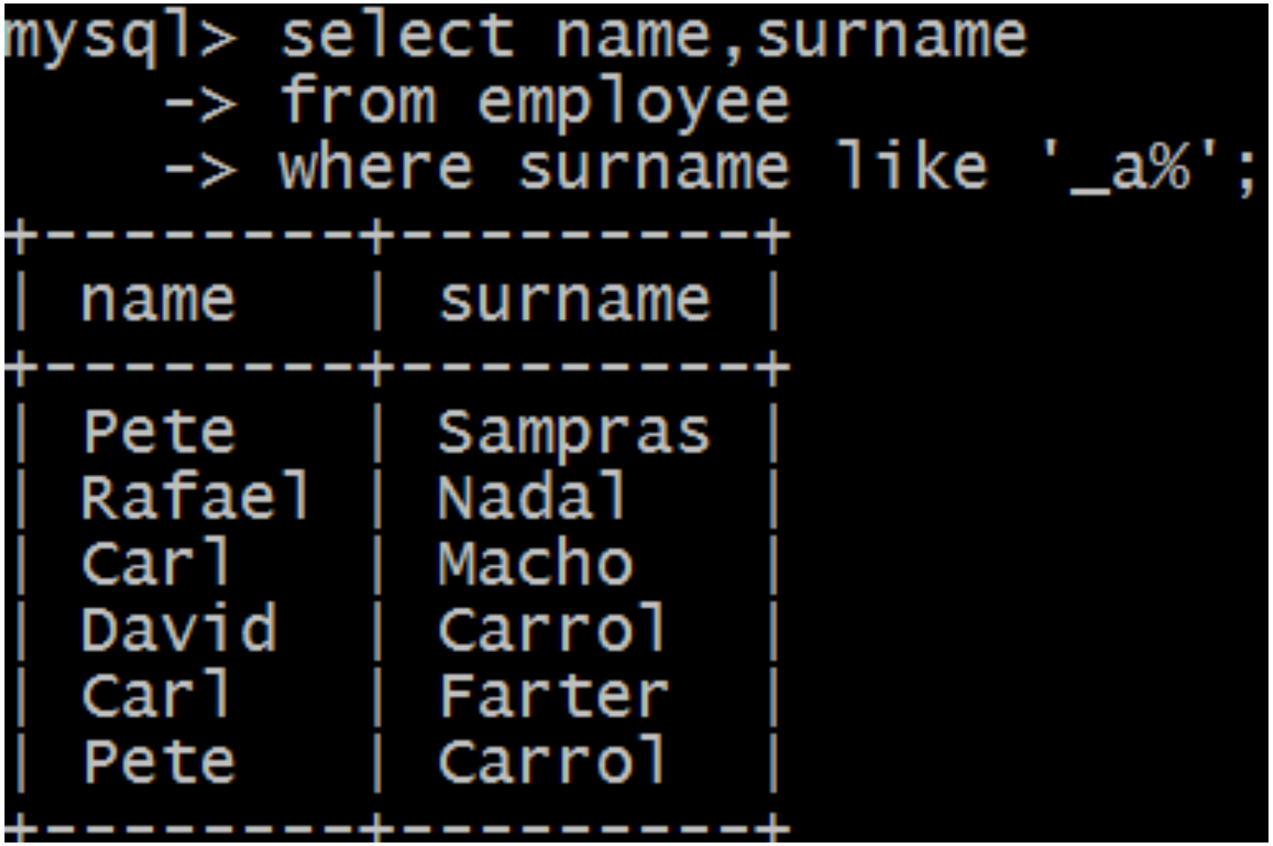
```
SELECT customer_name
FROM customer
WHERE customer_street LIKE "%Main%" ;
```

- **Eg:** Find the names of all customers whose street starts with the substring "Main%":

```
SELECT customer_name
FROM customer
WHERE customer_street LIKE "Main%";
```

String Operations (cont.)

- **Eg:** Find all Employees whose surname can start and end with any alphabet but it must have a letter ‘a’ as the second alphabet.



```
mysql> select name,surname
    -> from employee
    -> where surname like '_a%';
+--------+---------+
| name   | surname |
+--------+---------+
| Pete   | Sampras |
| Rafael | Nadal   |
| Carl   | Macho   |
| David  | Carrol  |
| Carl   | Farter  |
| Pete   | Carrol  |
+--------+---------+
```

Null values

- It is possible for tuples to have a **null value**, denoted by ***NULL***, for some of their attributes;
- ***NULL*** signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving NULL is NULL.
- Comparisons involving NULL return unknown:

UPPSALA UNIVERSITET

Null values (cont.)

- **Eg:** Find all loan numbers in the loan relation with NULL values for amount

```
SELECT loan_number FROM loan
WHERE amount IS NULL ;
```

- **Eg:** Find the total loan amounts:

```
SELECT SUM (amount) FROM loan ;
```

- The query ignores NULL amounts;
  - result is NULL if there is no non-null amount.
- All aggregate operations except count(*) ignore tuples with null values on the aggregated attributes.

Three-Valued Logic

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

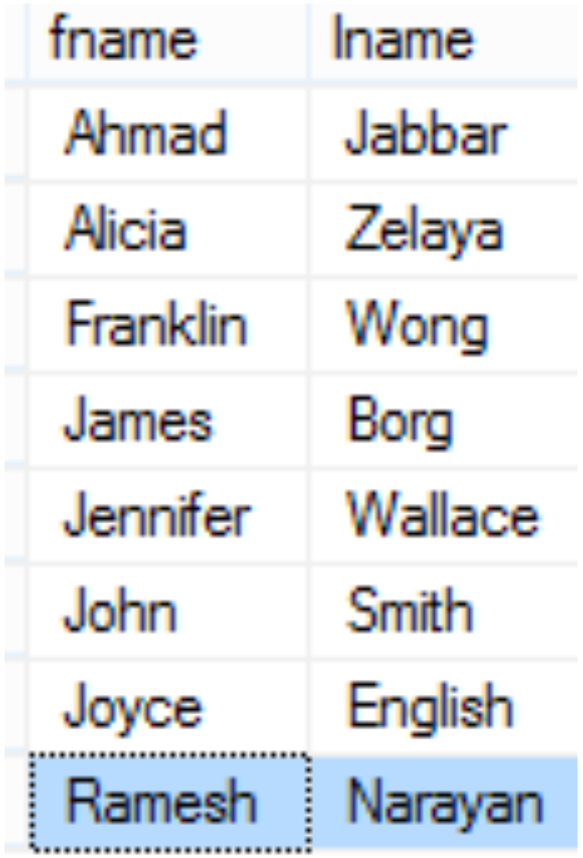
OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

Ordering the display of tuples

- SQL allows to order the tuples in the result of the query by the values of one or more attributes, using the **order by** clause.

Eg: List the names of employees ordered alphabetically by first name.

```
select fname, lname
from EMPLOYEE
order by fname
```



fname	lname
Ahmad	Jabbar
Alicia	Zelaya
Franklin	Wong
James	Borg
Jennifer	Wallace
John	Smith
Joyce	English
Ramesh	Narayan

Ordering the display of tuples (cont.)

We may specify **desc** for descending order or **asc** for ascending order, for each attribute; **ascending order is the default.**

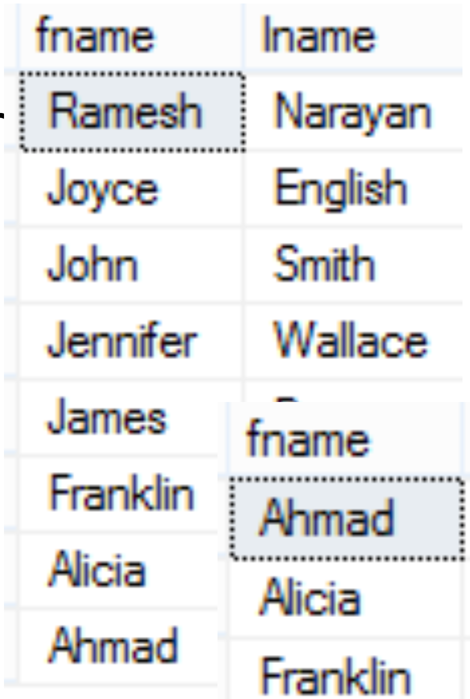
Eg:

```
select fname,lname
from EMPLOYEE
order by fname desc
```

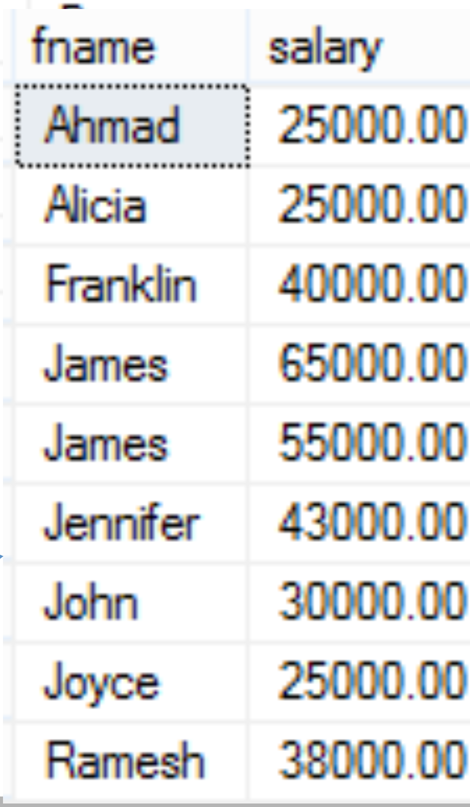
- Can sort on multiple attributes.

Eg:

```
select fname,salary
from employee
order by fname, salary desc;
```



fname	lname
Ramesh	Narayan
Joyce	English
John	Smith
Jennifer	Wallace
James	
Franklin	
Alicia	
Ahmad	



fname	salary
Ahmad	25000.00
Alicia	25000.00
Franklin	40000.00
James	65000.00
James	55000.00
Jennifer	43000.00
John	30000.00
Joyce	25000.00
Ramesh	38000.00

Set operations

- The set operations **UNION**, **INTERSECT**, and **EXCEPT** operate on relations and correspond to the set operators $\cup$, $\cap$, and $\setminus$ (sometimes written $-$).
- Each of the above operations automatically eliminates duplicates; to retain all duplicates use the corresponding multiset (sets with duplicates) versions **UNION ALL**, **INTERSECT ALL** and **EXCEPT ALL**.
- Suppose a tuple occurs *m* times in *r* and *n* times in *s*, then, it occurs:
  - m + n times in *r* **union all** *s*
  - min(m, n) times in *r* **intersect all** *s*
  - max(0, m - n) times in *r* **except all** *s*

Set operators

```
select <attribute_list>
from <list_of_tables>
where <predicate>
```

UNION [ALL] or INTERSECT [ALL] or EXCEPT [ALL]

```
select <attribute_list>
from <list_of_tables>
where <predicate>
```

Set operations: examples

- Find all customers who have a loan, an account, or both:

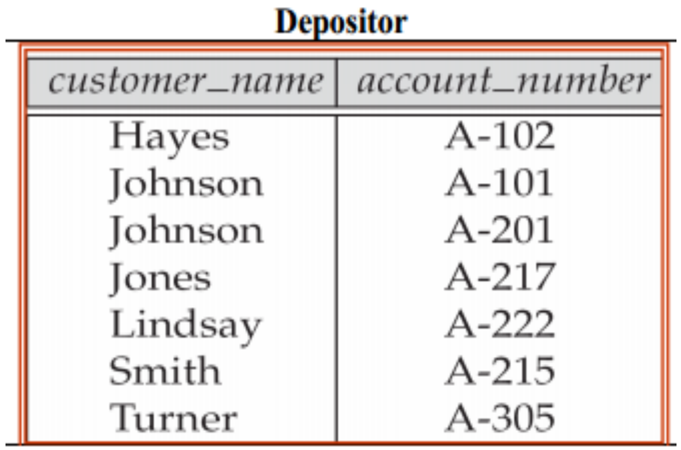
```
(SELECT customer_name FROM depositor)
UNION
(SELECT customer_name FROM borrower)
```

- Find all customers who have both a loan and an account:

```
(SELECT customer_name FROM depositor)
INTERSECT
(SELECT customer_name FROM borrower)
```

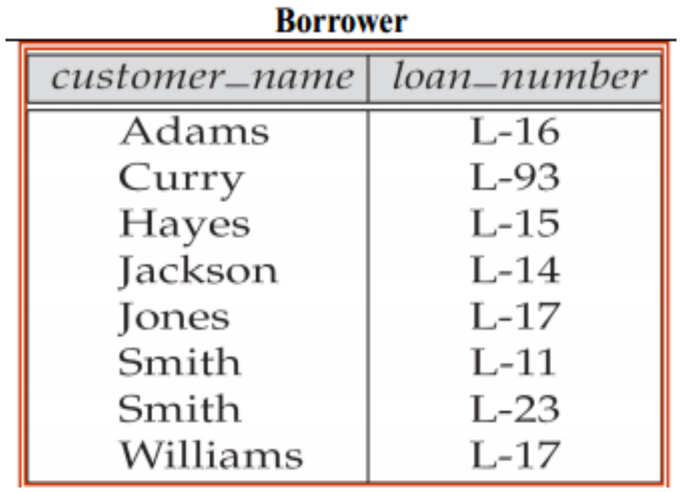
- Find all customers who have an account but no loan:

```
(SELECT customer_name FROM depositor)
EXCEPT
(SELECT customer_name FROM borrower)
```



Depositor

customer_name	*account_number*
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305



Borrower

customer_name	*loan_number*
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

AGGREGATE FUNCTIONS

Aggregate functions

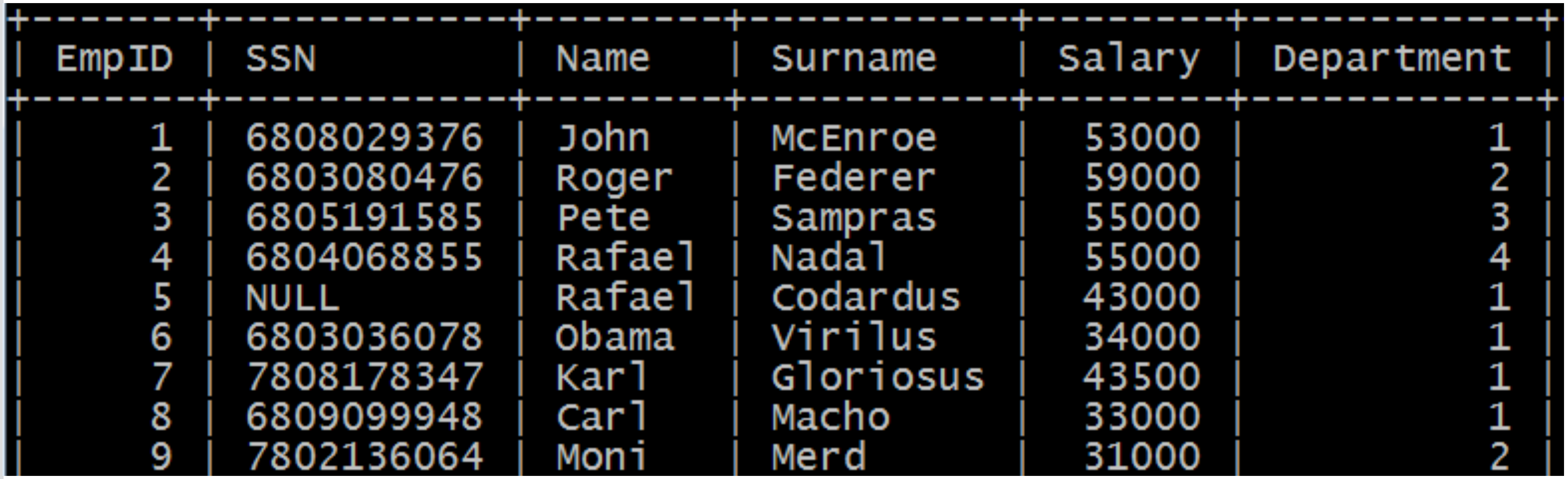
- In the previous queries predicates were verified on each single tuple.
- For example: get all employees whose salary is > than 30.
- Some advanced operations may address sets of tuples.
- For example: how many employees have a salary > than 30?
- SQL provides this functionality through aggregate functions.

```
select *
from Employee
where Salary > 30
```

Name	*Surname*	*Department*	*Supervisor*	*Salary*
John	White	1	2	36
Mark	Frank	1	3	46
~~Mean~~	~~Jones~~	~~2~~	~~1~~	~~27~~

Example

- Select the number of employees working at Department number 1.



```
+-------+------------+--------+-----------+--------+------------+
| EmpID | SSN        | Name   | Surname   | Salary | Department |
+-------+------------+--------+-----------+--------+------------+
|     1 | 6808029376 | John   | McEnroe   |  53000 |          1 |
|     2 | 6803080476 | Roger  | Federer   |  59000 |          2 |
|     3 | 6805191585 | Pete   | Sampras   |  55000 |          3 |
|     4 | 6804068855 | Rafael | Nadal     |  55000 |          4 |
|     5 | NULL       | Rafael | Codardus  |  43000 |          1 |
|     6 | 6803036078 | Obama  | Virilus   |  34000 |          1 |
|     7 | 7808178347 | Karl   | Gloriosus |  43500 |          1 |
|     8 | 6809099948 | Carl   | Macho     |  33000 |          1 |
|     9 | 7802136064 | Moni   | Merd      |  31000 |          2 |
```

UPPSALA UNIVERSITET

Evaluating aggregate queries (1)

```
select          *
from Employee
where Department = 1
```

Name	*Surname*	*Department*	*Supervisor*	*Salary*
John	White	1	2	36
Mark	Frank	1	3	46

UPPSALA UNIVERSITET

Evaluating aggregate queries (2)

```
select count(*) AS numberOfEmployees
from Employee
where Department = 1
```

Name	*Surname*	*Department*	*Supervisor*	*Salary*
John	White	1	2	36
Mark	Frank	1	3	46

numberOfEmployees
2

Other standard aggregate functions

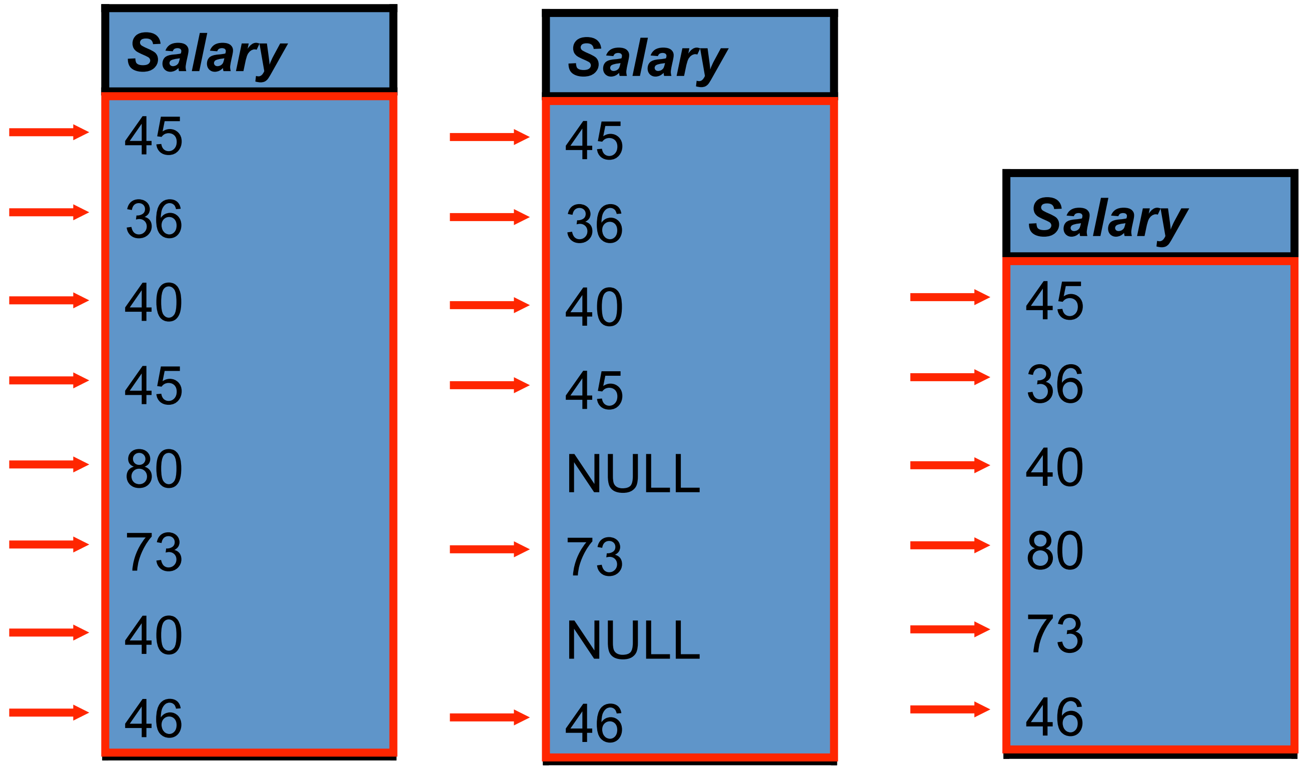
- count, sum, max, min, avg.
- Check the manual of the system you want to use for other options.

Target of the aggregate function

1) select count(*) AS numOfEmp from Employee

2) select count(salary) AS numOfSalaries from Employee

3) select count(distinct salary) AS numOfDistinctSalaries from Employee



	Salary
→	45
→	36
→	40
→	45
→	80
→	73
→	40
→	46

	Salary
→	45
→	36
→	40
→	45
	NULL
→	73
	NULL
→	46

	Salary
→	45
→	36
→	40
→	80
→	73
→	46

GROUP BY

- Aggregate functions work on groups of tuples.
- Instead of a single group (as in the previous example), we may want to apply an aggregate function to multiple groups of tuples inside the same relation.
- For example, count the aggregate salary of all employees **for each department**.

Sum of salaries per department

```
select Department,        Salary
from Employee
```

Department	*Salary*
1	45
2	36
1	40
3	45
4	80
4	73
1	40
2	46

Sum of salaries per department

```
select Department,        Salary
from Employee
GROUP BY Department
```

Department	*Salary*
1	45
1	40
1	40
2	36
2	46
3	45
4	80
4	73

Sum of salaries per department

```
select Department, sum(Salary)
from Employee
GROUP BY Department
```

Department	*Salary*
1	45
1	40
1	40
2	36
2	46
3	45
4	80
4	73

Department	
1	125
2	82
3	45
4	153

Sum of salaries per department

```
select Department, sum(Salary) AS allSalary
from Employee
GROUP BY Department
```

Department	*allSalary*
1	125
2	82
3	45
4	153

Aggregate functions and target list

- Only attributes used in the GROUP BY clause can appear in the target list outside aggregate functions (although some systems may allow it).
- For instance, the following query is not syntactically correct:

```
select D.Number, D.Name, count(*)
from Employee E, Department D
where E.Department = D.Number
group by D.Number
```

UPPSALA UNIVERSITET

Predicates on groups

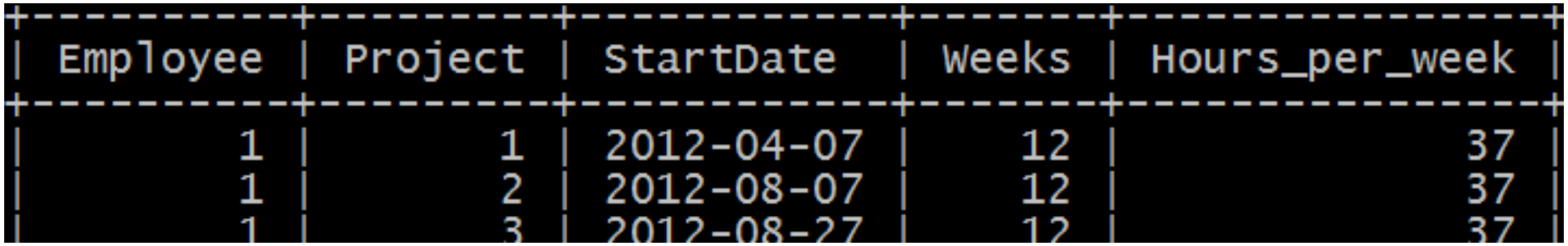
- Predicates involving aggregate functions are expressed in the HAVING clause.

```
select Department, sum(Salary)
from Employee
group by Department
HAVING sum(Salary) > 100
```

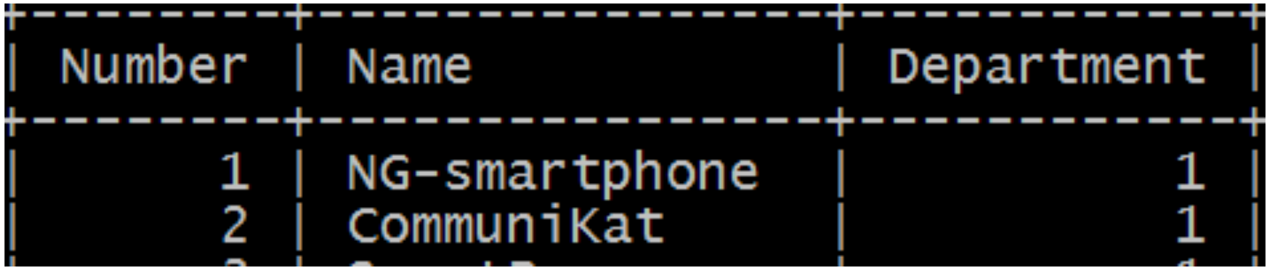
Predicates on groups (cont.)

- Retrieve the names of the project where only 3 employees work.

Works on



```
+----------+---------+------------+--------+----------------+
| Employee | Project | StartDate  | Weeks  | Hours_per_week |
+----------+---------+------------+--------+----------------+
|        1 |       1 | 2012-04-07 |     12 |             37 |
|        1 |       2 | 2012-08-07 |     12 |             37 |
|        1 |       3 | 2012-08-27 |     12 |             37 |
```



```
+--------+----------------+------------+
| Number | Name           | Department |
+--------+----------------+------------+
|      1 | NG-smartphone  |          1 |
|      2 | CommuniKat     |          1 |
```

project

select w.project,p.name,**count(w.employee)** 'No of Employees'
from works_on w join project p on w.project=p.number
group by w.project
having count(w.employee)=3

Wrap up

select target_list

from table_list

[where tuple_predicates]

[group by attribute_list]

[having group_predicates]

[order by attribute_list + ASC/DESC]

UPPSALA UNIVERSITET

Wrap up (cont.)

- Extensions of SELECT-FROM-WHERE.
  - LIKE, BETWEEN, IS (NOT) NULL.
  - DISTINCT.

- Set operators
  - UNION, EXCEPT, INTERSECT
- Aggregate functions: **count, max,min,avg, sum**