4. SQL **SELECT** Statement with clauses **GROUP BY** and **HAVING**

SELECT .... *columns names, expressions, functions (separated by a comma)*
FROM .... *tables or views names, joining clauses*
WHERE .... *the condition used to filter records*
GROUP BY .... *columns names, according to which the result set will be grouped*
HAVING .... *the condition used to filter groups*
ORDER BY .... *columns (or expressions) the result set is sorted by*

a) *Aliases to column names, expressions and tables*
An alias I used as a temporary name of a table or a column heading or to name an expression.
If an alias consists of two or more words, use quotation marks, e.g., "salary per month".

SELECT salary AS year_salary
FROM employee
	an alias for a column

An alias can be given to a table or view. Then in the SELECT statement we can use this alias to refer to this table

SELECT emp.last_name, emp.salary	*columns in employee are referred to using alias emp*
FROM employee emp	*table employee has alias emp*

We can refer to a table column in an SQL statement using

- *the name of the column:* *salary*
- *the name of the table and a column name, separated by a full stop:* employee.salary *(the table name is used as a qualifier for the column)*
- *an alias for a table and a column name, separated by a full stop:* *emp.salary* *(the alias is a qualifier)*

b) *some SQL* *scalar functions* *(SQL scalar functions return a single value, based on the input value)*

- *the function* UPPER(text) *returns the text with all letters changed to capital ones*
- *the function* LOWER(text) *returns the text with all letters changed to small ones*
- *the function* CAST(data AS data_type) *is used to make a conversion from one data type to another*

```
SELECT UPPER(full_name), CAST(salary AS CHAR(12))
FROM employee
```

c) ***SQL aggregate functions***

SQL aggregate functions return a single value, calculated from values in a column. Most useful are the following:

▪ **COUNT**(*)	*– returns the number of rows that matches a specified criteria (counts the number of rows)*	
▪ **SUM**(column_name)	*- returns the total sum of a given column*	*//only for numeric columns*
▪ **AVG**(column_name)	*- returns the arithmetic average value of a given column*	*// only for numeric columns*
▪ **MIN**(column_name)	*- returns the smallest value of the given column*	
▪ **MAX**(column_name)	*- returns the largest value of the given column*	

- *If as an argument of an aggregating function we put* DISTINCT column_name, *then the function will take into account only distinct values from a given column.*
- COUNT(column) *- returns the number of non-empty values in the given column*
- COUNT(DISTINCT column) *- returns the number of distinct non-empty values in the specified column*
- *An aggregate function cannot be used in WHERE clause*

```
SELECT COUNT(*), SUM(SALARY)
FROM employee
WHERE dept_no='600'

SELECT AVG(DISTINCT salary)
FROM employee
```

```
SELECT dept_no, COUNT(phone_ext)
FROM employee
GROUP BY dept_no

SELECT MIN(LOWER(job_code)), dept_no
FROM employee
GROUP BY dept_no
```

d) **GROUP BY** *clause is used to group the result-set by one or more columns*

```
SELECT …
FROM …
GROUP BY column_name
```

- *If we use the group by clause in the SELECT statement, then one group will consists of all rows, having the same values in the columns specified in GROUP BY*
- GROUP BY *is often used with aggregate functions. In such a case, a value of an aggregate function is computed for each group separately*
- *If we use GROUP BY, then on a SELECT list we can put only:*
  - *Names of columns, which are in the GROUP BY clause,*

        - *Expressions that use only the columns, which are in the GROUP BY clause*
        - *Constants*
        - *Aggregate functions*
    - *If we use GROUP BY, then in an ORDER BY clause we may put only the columns, which are used in GROUP BY*

```
SELECT COUNT(*), AVG(SALARY), dept_no
FROM employee
GROUP BY dept_no
```

```
SELECT COUNT(DISTINCT dept_no), SUM(SALARY), job_country
FROM employee
GROUP BY job_country
ORDER BY SUM(SALARY)
```

```
SELECT SUM(SALARY), job_country, dept_no
FROM employee
GROUP BY job_country, dept_no
ORDER BY SUM(SALARY)
```

```
SELECT SUM(SALARY), job_country, dept_no
FROM employee
WHERE hire_date>=’1990.12.23’
GROUP BY job_country, dept_no
ORDER BY SUM(SALARY) DESC
```

e) **HAVING** *clause is used to specify a criterium for choosing groups. We put there conditions, which are used to decide which groups should be included in the result set. In this conditions we can use only the columns or expressions, which are in the GROUP BY clause.*
    - *HAVING can be used only together with the GROUP BY clause*
    - *In the conditions put in the **HAVING** clause, we can use aggregate functions (it is in fact the main purpose for using HAVING)*
    - *All conditions referring only to **rows**, but **not to groups**, should be put in the WHERE clause*

```
SELECT dept_no, SUM(salary)
FROM employee
GROUP BY dept_no
HAVING COUNT(*)<=3
```

```
SELECT dept_no, COUNT(*), AVG(SALARY)
FROM employee
WHERE hire_date>=’1999.09.18’
GROUP BY dept_no
HAVING SUM(salary)>=30000 AND COUNT(*)>=3
ORDER BY AVG(salary) DESC
```