**CREATE** **TABLE** Employee

(

ID **INT** **NOT** **null** **GENERATED** **ALWAYS** **AS** **IDENTITY**,

"Name" **VARCHAR**(100) **not** **null**,

Salary **int** **not** **null**,

ID\_Departament **int** **not** **null**,

**PRIMARY** **KEY** (ID),

**FOREIGN** **KEY** (ID\_Departament) **REFERENCES** Departament(ID\_Departament)

);

**create** **table** Departament(

ID\_Departament **int** **not** **null** **GENERATED** **ALWAYS** **AS** **IDENTITY**,

"Name" **VARCHAR**(100) **not** **null**,

**PRIMARY** **KEY** (ID\_Departament)

);

**INSERT** **INTO** Departament("Name")

**VALUES** ('Sales'),

('Marketing'),

('Human Resources'),

('Finance'),

('IT');

**INSERT** **INTO** Employee("Name", Salary, ID\_Departament)

**VALUES** ('Jane Smith', 60000, 2),

('Alex Johnson', 70000, 2),

('Emily Davis', 55000, 1),

('Mike Brown', 65000, 3),

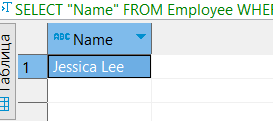
('Jessica Lee', 75000, 3);

1. Сотрудника с максимальной заработной платой.

**SELECT** "Name"

**FROM** Employee

**WHERE** Salary = (**SELECT** **MAX**(Salary) **FROM** Employee);



Вывести одно число: максимальную длину цепочки руководителей по таблице сотрудников (вычислить глубину дерева). **WITH** **RECURSIVE** manager\_tree **AS** (

**SELECT** ID, chief\_id, 1 **AS** **depth**

**FROM** Employee

**WHERE** chief\_id **IS** **NULL**

**UNION** **ALL**

**SELECT** e.ID, e.chief\_id, mt.**depth** + 1 **AS** **depth**

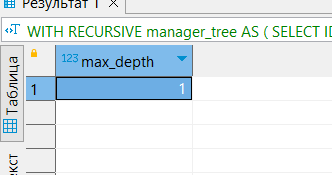
**FROM** Employee e

**JOIN** manager\_tree mt **ON** e.chief\_id = mt.ID

)

**SELECT** **MAX**(**depth**) **AS** max\_depth

**FROM** manager\_tree;



Отдел, с максимальной суммарной зарплатой сотрудников.

**SELECT** d."Name", **SUM**(e.Salary) **AS** total\_salary

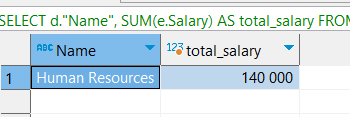
**FROM** Employee e

**JOIN** Departament d **ON** e.ID\_Departament = d.ID\_Departament

**GROUP** **BY** d."Name"

**ORDER** **BY** total\_salary **DESC**

**LIMIT** 1;

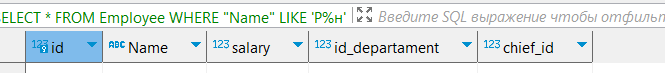


Сотрудника, чье имя начинается на «Р» и заканчивается на «н».

**SELECT** \*

**FROM** Employee

**WHERE** "Name" **LIKE** 'Р%н';



using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

namespace WordCount

{

class Program

{

static void Main(string[] args)

{

if (args.Length != 2)

{

Console.WriteLine("Usage: WordCount <input\_file> <output\_file>");

return;

}

string inputFilePath = args[0];

string outputFilePath = args[1];

try

{

// Read input file into a list of words

List<string> words = new List<string>();

using (StreamReader sr = new StreamReader(inputFilePath))

{

string line;

while ((line = sr.ReadLine()) != null)

{

string[] lineWords = line.Split(' ', '\t');

words.AddRange(lineWords);

}

}

// Count occurrences of each word

Dictionary<string, int> wordCounts = new Dictionary<string, int>();

foreach (string word in words)

{

if (string.IsNullOrWhiteSpace(word))

{

continue;

}

string cleanedWord = word.Trim().ToLower();

if (wordCounts.ContainsKey(cleanedWord))

{

wordCounts[cleanedWord]++;

}

else

{

wordCounts[cleanedWord] = 1;

}

}

// Sort words by count

List<KeyValuePair<string, int>> sortedWords = wordCounts.ToList();

sortedWords.Sort((x, y) => y.Value.CompareTo(x.Value));

// Write sorted word counts to output file

using (StreamWriter sw = new StreamWriter(outputFilePath))

{

foreach (KeyValuePair<string, int> wordCount in sortedWords)

{

sw.WriteLine($"{wordCount.Key}\t\t{wordCount.Value}");

}

}

}

catch (Exception e)

{

Console.WriteLine($"Error: {e.Message}");

}

}

}

}

WordCount.exe input.txt output.txt