Indices and Data Aggregation

How to Get Data Insights?

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Table of Contents

Questions

Software University

sli.do

#csharp-db

Indices

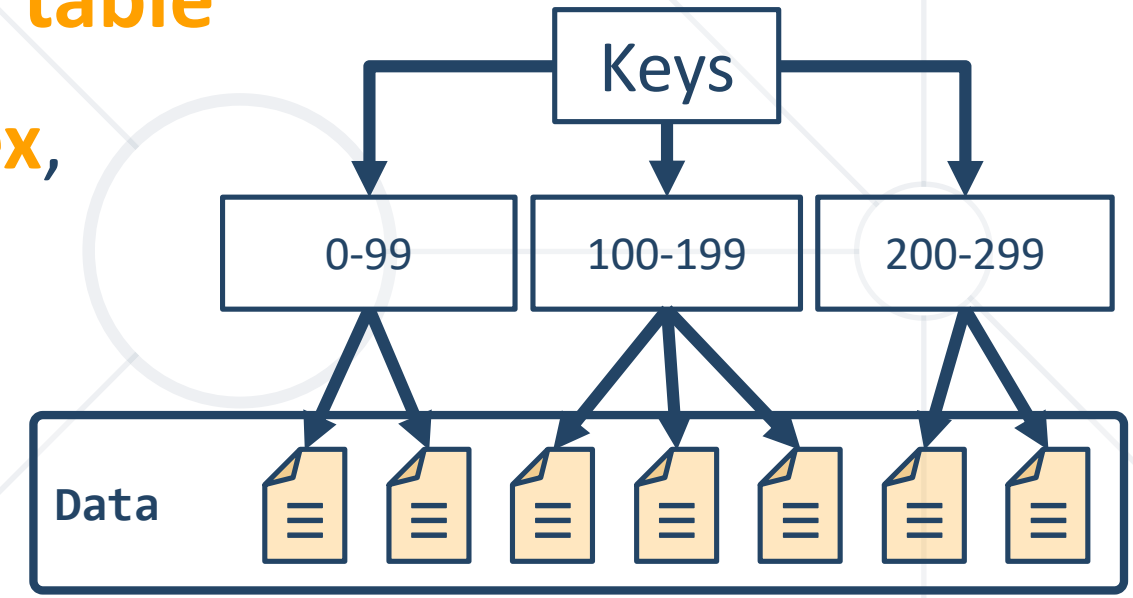
Clustered and Non-Clustered Indexes

Indices

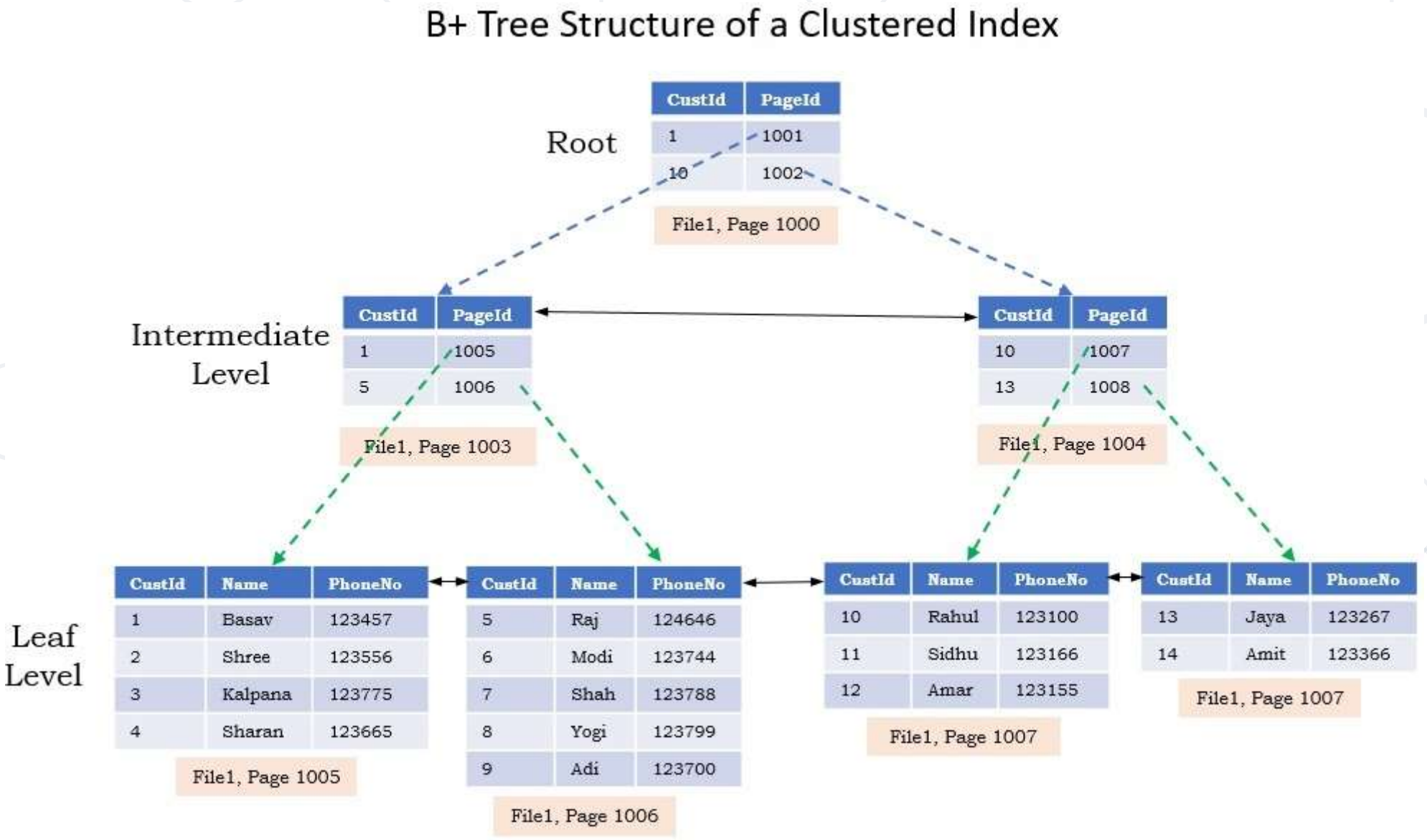
- **Indices speed up the searching of values** in a certain column or group of columns
  - Usually implemented as **B-trees**
- Indices can be **built-in the table** (**clustered**) or **stored externally** (**non-clustered**)
- **Adding** and **deleting** records in indexed tables is **slower**!
  - Indices should be used **for big tables only** (e.g. 500 000 rows).

Clustered Indices

- **Clustered index** is actually **the data itself**
  - Very useful for **fast execution** of **`WHERE`**, **`ORDER BY`** and **`GROUP BY`** clauses
- Maximum **1** clustered index **per table**
  - If a table **has no clustered index**, its **data rows are stored in** an unordered structure (**heap**).



Clustered Indexes (2)

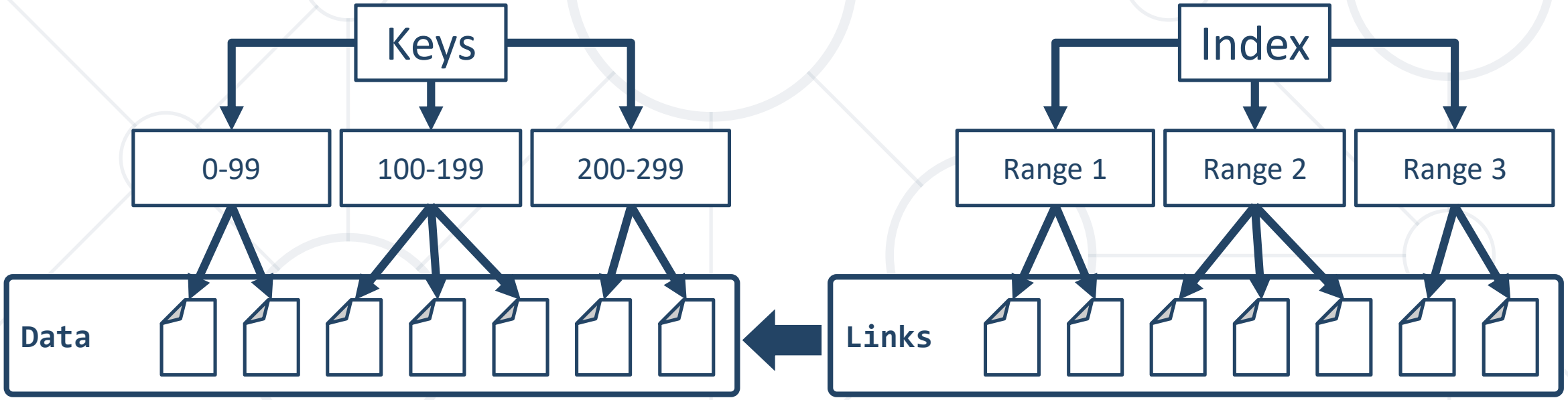


Non-Clustered Indeces

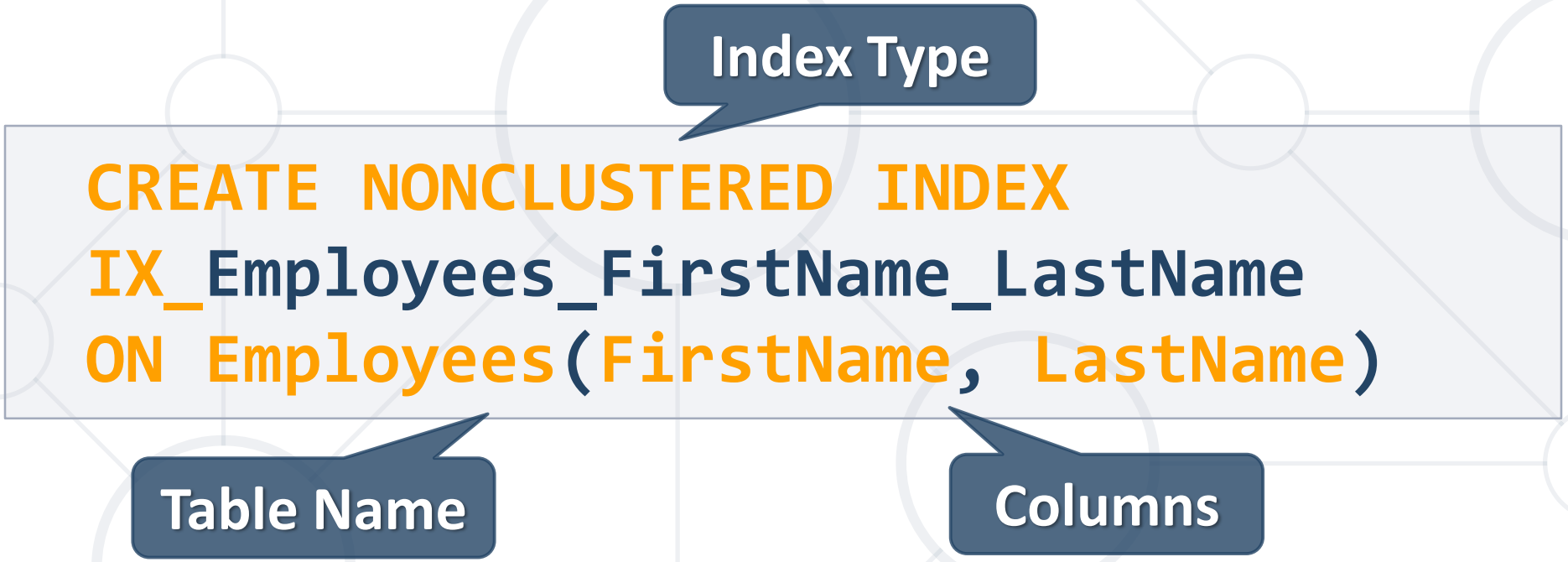
- Useful for **fast retrieving** of a range of records
- Maintained in a **separate structure** in the DB
- Tend to be **much narrower** than the base table
  - Can **locate the exact record(s)** with **less I/O**
- Has **at least one more intermediate level** than the clustered index
  - Much **less valuable** if a table **doesn't have a clustered index**

Non-Clustered Indexes (2)

- A non-clustered index **has pointers** to the **actual data rows** (pointers to the clustered index if there is one).



Indices Syntax



Demo: Index Performance

Live Demo

Grouping

Consolidating Data Based On Criteria

Grouping (1)

- **Grouping** allows receiving data into separate groups based on a common property

Grouping column

Employee	DepartmentName	Salary
Adam	Database Support	5,000
John	Database Support	15,000
Jane	Application Support	10,000
George	Application Support	15,000
Lila	Application Support	5,000
Fred	Software Support	15,000

Single row

Can be aggregated

Grouping (2)

- **GROUP BY** allows you to get each **separate group** and use an "**aggregate**" function over it (like **Average**, **Min** or **Max**):

```
   SELECT e.DepartmentID
     FROM Employees AS e
GROUP BY e.DepartmentID
```

Group Columns

- **DISTINCT** allows you to get **all unique** values:

```
SELECT DISTINCT e.DepartmentID
   FROM Employees AS e
```

Unique Values

Problem: Departments Total Salaries

- Use "**SoftUni**" **database** to create a query which prints the total sum of salaries for each department
  - Order them by **DepartmentID** (ascending)

Employee	DepartmentID	Salary
Adam	1	5,000
John	1	15,000
Jane	2	10,000
George	2	15,000
Lila	2	5,000
Fred	3	15,000

DepartmentID	TotalSalary
1	20,000
2	30,000
3	15,000

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/291#12

Solution: Departments Total Salaries

- After **grouping** every employee **by** its **department**, we can use an **aggregate function** to calculate the total amount of money per group

```
SELECT e.DepartmentID,
  SUM(e.Salary) AS TotalSalary
FROM Employees AS e
GROUP BY e.DepartmentID
ORDER BY e.DepartmentID
```

Column Alias

Table Alias

Group Columns

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/291#12

Aggregate Functions

COUNT, SUM, MAX, MIN, AVG…

Aggregate Functions

- Operate over (**non-empty**) **groups**
- Perform **data analysis** on each one
  - **MIN**, **MAX**, **AVG**, **COUNT**, etc.

```
SELECT e.DepartmentID,
 MIN(e.Salary) AS MinSalary
FROM Employees AS e
GROUP BY e.DepartmentID
```

DepartmentID	MinSalary
1	32700.00
2	25000.00
3	23100.00
4	13500.00
5	12800.00
6	40900.00
7	9500.00

- Aggregate functions usually **ignore NULL** values

Aggregate Functions: COUNT

- **COUNT** - **counts the values** in one or more **grouped columns**
  - **Ignores NULL** values

Employee	DepartmentName	Salary
Adam	Database Support	5,000
John	Database Support	15,000
Jane	Application Support	10,000
George	Application Support	15,000
Lila	Application Support	5,000
Fred	Software Support	15,000

DepartmentName	SalaryCount
Database Support	2
Application Support	3
Software Support	1

COUNT Syntax

- `COUNT(ColumnName)`

```
   SELECT e.DepartmentID,
          COUNT(e.Salary) AS SalaryCount
     FROM Employees AS e
GROUP BY e.DepartmentID
```

New Column Alias

Group Columns

- Note: **COUNT ignores** any employee with **NULL** salary

Aggregate Functions: SUM

- **SUM** - **sums the values** in a column

Employee	DepartmentName	Salary
Adam	Database Support	5,000
John	Database Support	15,000
Jane	Application Support	10,000
George	Application Support	15,000
Lila	Application Support	5,000
Fred	Software Support	15,000

DepartmentName	TotalSalary
Database Support	20,000
Application Support	30,000
Software Support	15,000

SUM Syntax

- If any department **has no salaries**, it **returns NULL**


```
   SELECT e.DepartmentID,
          SUM(e.Salary) AS TotalSalary
     FROM Employees AS e
GROUP BY e.DepartmentID
```

Aggregate Functions: MAX

- **MAX** - takes **the largest value** in a column

Employee	DepartmentName	Salary
Adam	Database Support	5,000
John	Database Support	15,000
Jane	Application Support	10,000
George	Application Support	15,000
Lila	Application Support	5,000
Fred	Software Support	15,000

DepartmentName	MaxSalary
Database Support	15,000
Application Support	15,000
Software Support	15,000

MAX Syntax

Grouping Column

New Column Alias

```
   SELECT e.DepartmentID,
          MAX(e.Salary) AS MaxSalary
     FROM Employees AS e
GROUP BY e.DepartmentID
```

Group Columns

Aggregate Functions: MIN

- `MIN` - takes **the smallest value** in a column

Employee	DepartmentName	Salary
Adam	Database Support	5,000
John	Database Support	15,000
Jane	Application Support	10,000
George	Application Support	15,000
Lila	Application Support	5,000
Fred	Software Support	15,000

DepartmentName	MinSalary
Database Support	5,000
Application Support	5,000
Software Support	15,000

MIN Syntax

```
   SELECT e.DepartmentID,
          MIN(e.Salary) AS MinSalary
     FROM Employees AS e
GROUP BY e.DepartmentID
```

New Column Alias

Group Columns

Aggregate Functions: AVG

- **AVG** - calculates the **average value** in a column

Employee	DepartmentName	Salary
Adam	Database Support	5,000
John	Database Support	15,000
Jane	Application Support	10,000
George	Application Support	15,000
Lila	Application Support	5,000
Fred	Software Support	15,000

DepartmentName	AvgSalary
Database Support	10,000
Application Support	10,000
Software Support	15,000

AVG Syntax

```
   SELECT e.DepartmentID,
          AVG(e.Salary) AS AvgSalary
     FROM Employees AS e
GROUP BY e.DepartmentID
```

New Column Alias

Group Columns

Aggregate Functions: STRING_AGG

- **STRING_AGG** - Concatenates the values of string expressions and places separator values between them. The separator is not added at the end of string

Expressions are converted to NVARCHAR or VARCHAR types during concatenation. Non-string types are converted to NVARCHAR type

```
STRING_AGG ( expression, separator )
  [WITHIN GROUP ( ORDER BY expression [ ASC | DESC ] )]
```

Having

Using Predicates While Grouping

HAVING Clause

- The **HAVING clause** is used to **filter data** based on **aggregate values**
  - We **cannot** use it **without grouping** first
- **Aggregate functions** (MIN, MAX, SUM etc.) are **executed only once**
  - Unlike HAVING, **WHERE filters** rows **before aggregation**

HAVING Clause: Example

- Filter departments having total salary more than or equal to 15,000

Aggregated value

Employee	DepartmentName	Salary	TotalSalary
Adam	Database Support	5,000	20,000
John	Database Support	15,000	
Jane	Application Support	1,000	11,000
George	Application Support	5,000	
Lila	Application Support	5,000	
Fred	Software Support	15,000	15,000

DepartmentName	TotalSalary
Database Support	20,000
Software Support	15,000

HAVING Syntax

```
   SELECT e.DepartmentID,
          SUM(e.Salary) AS TotalSalary
     FROM Employees AS e
GROUP BY e.DepartmentID
  HAVING SUM(e.Salary) >= 15000
```

Aggregate Function

Column Alias

Grouping Columns

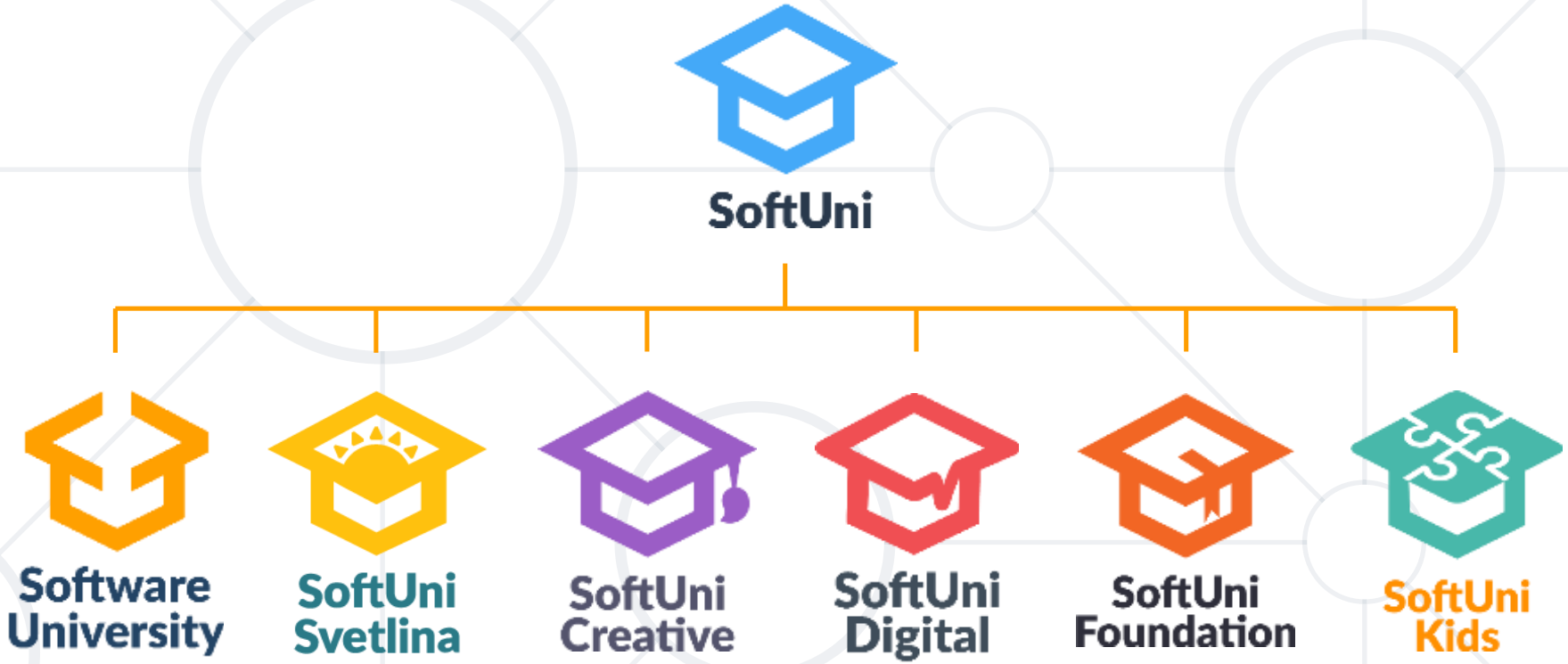
Having Predicate

Summary

- Grouping by Shared Properties
- Aggregate Functions
- Having Clause

```
SELECT
  SUM(e.Salary) AS 'TotalSalary'
FROM Employees AS e
GROUP BY e.DepartmentID
HAVING SUM(e.Salary) >= 15_000
```

Questions?

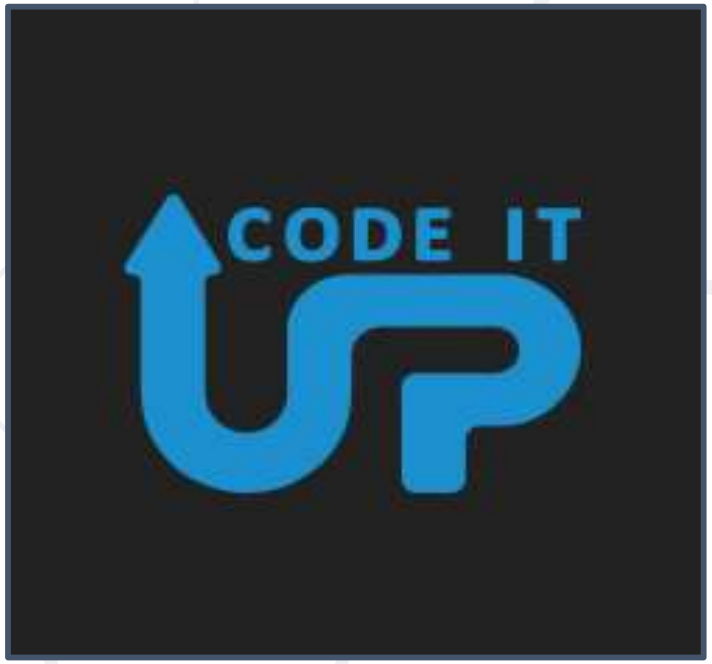


SoftUni Diamond Partners

Software University

Educational Partners

Software University



VIRTUAL RACING SCHOOL

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg
- © Software University – https://softuni.bg