Joins, Subqueries, CTEs and Indexes

Table of Content

1. Joins
2. Subqueries
3. Common Table Expressions (CTE)
4. Indexes

JOINS

Gathering Data From Multiple Tables

Data from Multiple Tables

- Sometimes you need data from several tables:

Employees

EmployeeName	DepartmentID
Edward	3
John	4

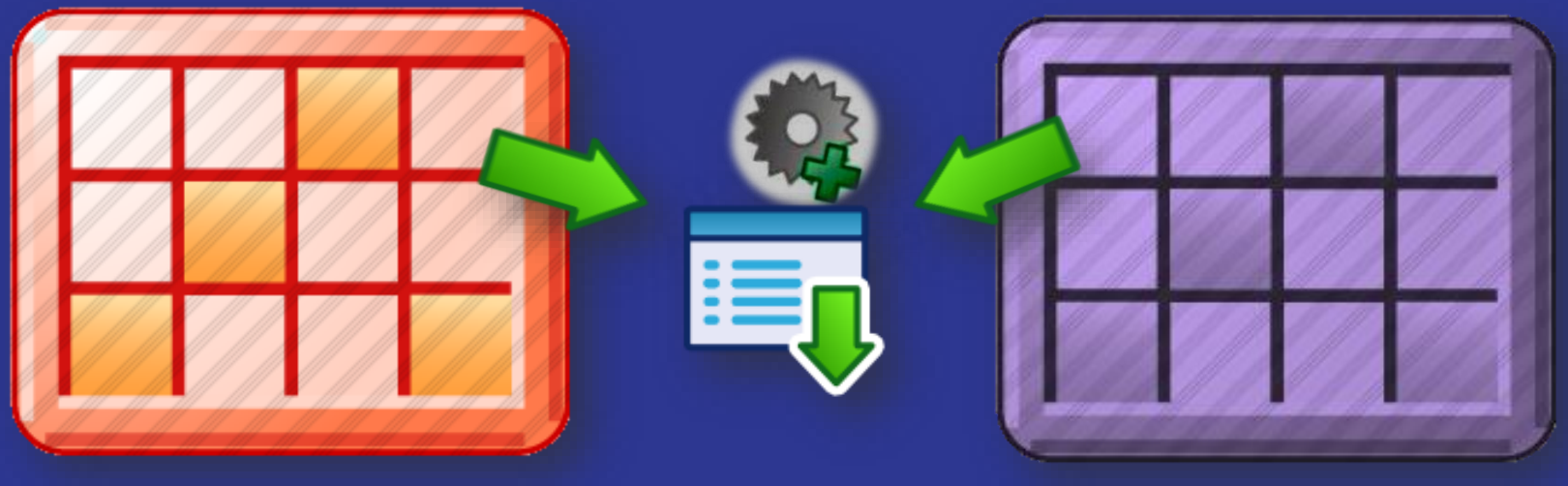
Departments

DepartmentID	DepartmentName
3	Sales
4	Marketing
5	Purchasing

EmployeeName	DepartmentID	DepartmentName
Edward	3	Sales
John	4	Marketing

Types of Joins

- Inner joins
- Left, right and full outer joins
- Cross joins



INNER vs. OUTER Joins

- **Inner join**
  - A join of two tables returning only rows matching the join condition
- **Left (or right) outer join**
  - Returns the results of the inner join as well as unmatched rows from the left (or right) table
- **Full outer join**
  - Returns the results of an inner join along with all unmatched rows

Inner Join

Employees

EmployeeID	DepartmentID
263	3
270	4

Departments

DepartmentID	DepartmentName
3	Sales
4	Marketing
5	Purchasing

Result

EmployeeID	DepartmentID	DepartmentID	DepartmentName
263	3	3	Sales
270	4	4	Marketing

Inner Join Syntax

```
SELECT * FROM Employees AS e
  INNER JOIN Departments AS d
  ON e.DepartmentID = d.DepartmentID
```

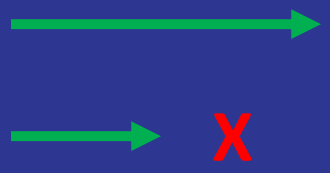
Depatments Table

Join Condition

Left Outer Join

Employees

EmployeeID	DepartmentID
263	3
270	NULL



Departments

DepartmentID	DepartmentName
3	Sales
4	Marketing
5	Purchasing

Result

EmployeeID	DepartmentID	DepartmentID	DepartmentName
263	3	3	Sales
270	NULL	NULL	NULL

Left Outer Join Syntax

```
SELECT * FROM Employees AS e
  LEFT OUTER JOIN Departments AS d
  ON e.DepartmentID = d.DepartmentID
```

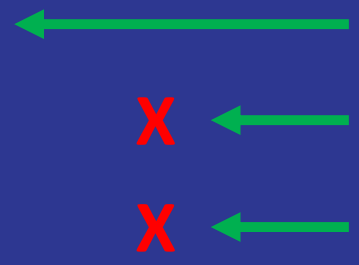
Table Depatments

Join Condition

Right Outer Join

Employees

EmployeeID	DepartmentID
263	3
270	NULL



Departments

DepartmentID	DepartmentName
3	Sales
4	Marketing
5	Purchasing

Result

EmployeeID	DepartmentID	DepartmentID	DepartmentName
263	3	3	Sales
NULL	NULL	4	Marketing
NULL	NULL	5	Purchasing

Right Outer Join Syntax

```
SELECT * FROM Employees AS e
 RIGHT OUTER JOIN Departments AS d
 ON e.DepartmentID = d.DepartmentID
```

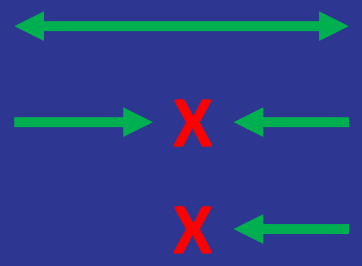
Depatments Table

Join Condition

Full Join

Employees

EmployeeID	DepartmentID
263	3
270	NULL



Departments

DepartmentID	DepartmentName
3	Sales
4	Marketing
5	Purchasing

Result

EmployeeID	DepartmentID	DepartmentID	DepartmentName
263	3	3	Sales
270	NULL	NULL	NULL
NULL	NULL	4	Marketing
NULL	NULL	5	Purchasing

Full Join Syntax

```
SELECT * FROM Employees AS e
  FULL JOIN Departments AS d
  ON e.DepartmentID = d.DepartmentID
```

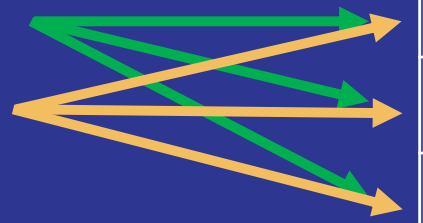
Depatments Table

Join Condition

Cross Join

Employees

EmployeeID	DepartmentID
263	3
270	NULL



Departments

DepartmentID	DepartmentName
3	Sales
4	Marketing
5	Purchasing

Result

EmployeeID	DepartmentID	DepartmentID	DepartmentName
263	3	3	Sales
263	3	4	Marketing
263	3	5	Purchasing
270	NULL	3	Sales
270	NULL	4	Marketing
270	NULL	5	Purchasing

Cross Join Syntax

```
SELECT * FROM Employees AS e
 CROSS JOIN Departments AS d
```

Depatments Table

No Join Conditions

Join Overview

Sally	13
John	10
Michael	22
Bob	11
Robin	7
Jessica	15

18	Accounting
10	Marketing
12	HR
22	Engineering
8	Sales
7	Executive

Relation

Join Overview

Sally	13
John	10

Michael	22

Bob	11
Robin	7
Jessica	15

18	Accounting

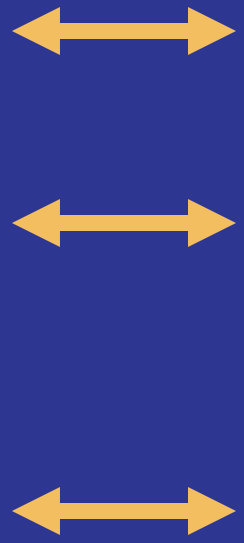
10	Marketing
12	HR
22	Engineering
8	Sales

7	Executive

Join Overview

- Inner Join

Sally	13
John	10
Michael	22
Bob	11
Robin	7
Jessica	15

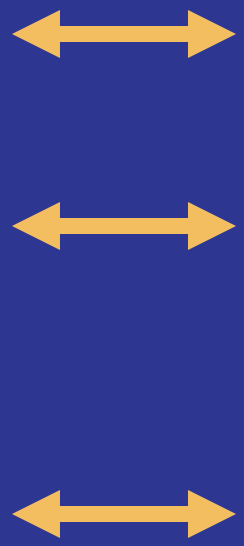


18	Accounting
10	Marketing
12	HR
22	Engineering
8	Sales
7	Executive

Join Overview

- Left Outer Join

Sally	13
John	10
Michael	22
Bob	11
Robin	7
Jessica	15

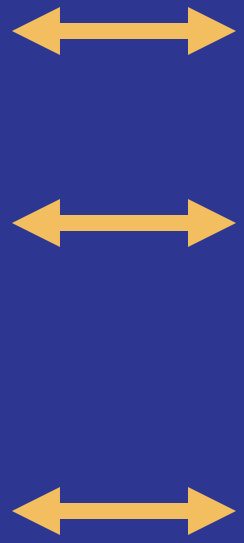


18	Accounting
NULL	*NULL*
10	Marketing
12	HR
22	Engineering
8	Sales
NULL	*NULL*
7	Executive
NULL	*NULL*

Join Overview

- Right Outer Join

NULL	*NULL*
Sally	13
John	10
NULL	*NULL*
Michael	22
NULL	*NULL*
Bob	11
Robin	7
Jessica	15

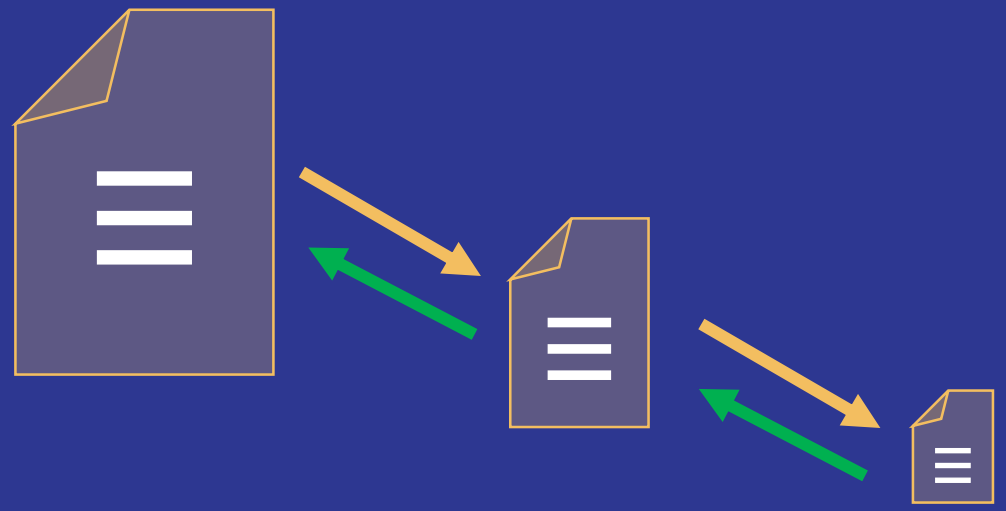


18	Accounting
10	Marketing
12	HR
22	Engineering
8	Sales
7	Executive

Join Overview

- Full Outer Join

NULL	*NULL*		18	Accounting
Sally	13		*NULL*	*NULL*
John	10	⟷	10	Marketing
NULL	*NULL*		12	HR
Michael	22	⟷	22	Engineering
NULL	*NULL*		8	Sales
Bob	11		*NULL*	*NULL*
Robin	7	⟷	7	Executive
Jessica	15		*NULL*	*NULL*



Subqueries

Query Manipulation on Multiple Levels

Subqueries

- Use a query's result as data for another query

Employees

EmployeeID	Salary
59	19,000
71	43,300
…	…

Query

```
WHERE DepartmentID IN
```

Subquery

DepartmentID	Name
10	Finance

Subquery Syntax

```
SELECT * FROM Employees AS e
 WHERE e.DepartmentID IN
   (
      SELECT d.DepartmentID
      FROM Deparments AS d
      WHERE d.Name = 'Finance'
   )
```

Operator

Table Depatments

Subquery

Restrictions with Operators

- Use a query's result as data for another query

	One Column	Many Columns
One Row	Use =, <, > and other comparison operators	Use EXISTS
Many Rows	Use ANY, ALL, IN and EXISTS	Use EXISTS

```
SELECT * FROM EXAMMARK
 WHERE Marks > ANY
  (
     SELECT Marks
     FROM EXAMMARK
     WHERE Marks in (40,50,70)
  )
```

EXISTS keyword

- The EXISTS keyword is used with a subquery to check the existence of rows returned by the subquery.
- The subquery does not actually return any data; it returns a value of TRUE or FALSE.
- The syntax of a subquery containing the word EXISTS is as follows:

```
SELECT <Column Name> FROM <table>
WHERE [NOT] EXISTS
(
        <Subquery_Statement>
)
```

EXISTS keyword - Example

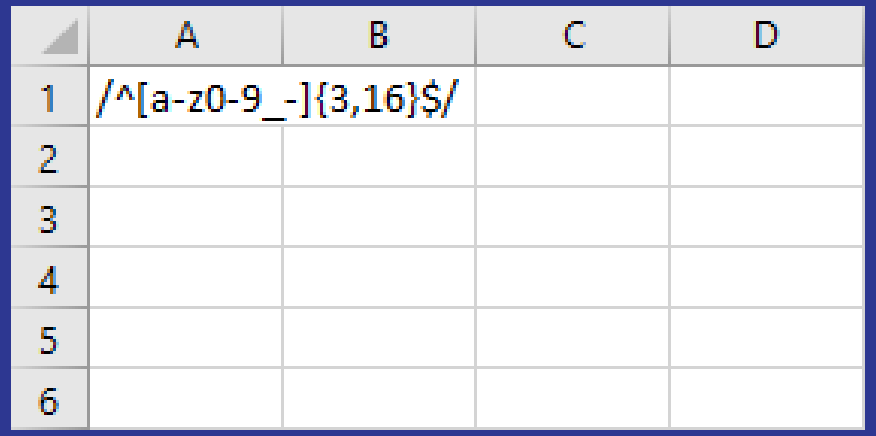
- Display Subjects that students have tested

```
SELECT * FROM SUBJECT AS S
 WHERE EXISTS
  (
      SELECT *
      FROM EXAMMARKS
      WHERE SUBID = S.SUBID
  )
```

Table SUBJECT

EXISTS

SUBQUERY



Common Table Expressions

Reusable Subqueries

Common Table Expressions

- Common Table Expressions (CTE) can be considered as "named subqueries".
- They could be used to improve code readability and code reuse.
- Usually they are positioned in the beginning of the query.

```
WITH CTE_Name (ColumnA, ColumnB…)
AS
(
  -- Insert subquery here.
)
```

CTE Syntax

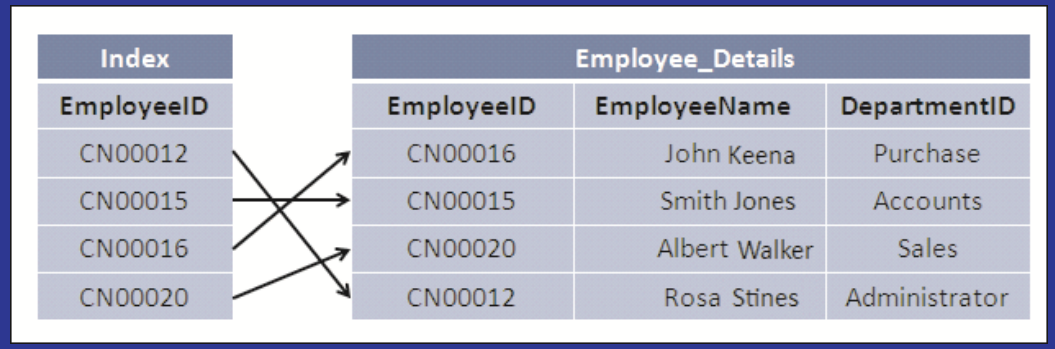
```
WITH Employees_CTE
  (FirstName, LastName, DepartmentName)
AS
(
  SELECT e.FirstName, e.LastName, d.Name
  FROM Employees AS e
  LEFT JOIN Departments AS d ON
    d.DepartmentID = e.DepartmentID
)

SELECT FirstName, LastName, DepartmentName
FROM Employees_CTE
```

Indexes

Clustered and Non-Clustered Indexes

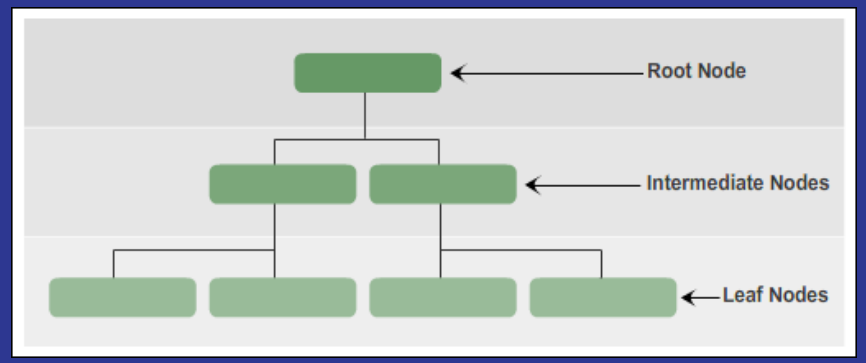
Indexes

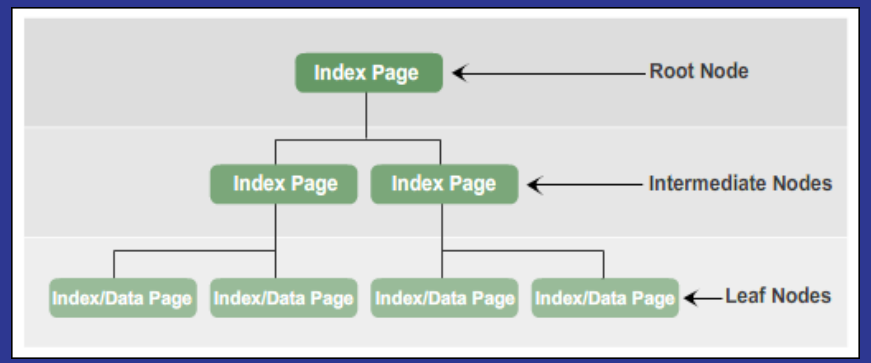
- In a table, records are stored in the order in which they are entered. Their storage in the database is unsorted.
- When data is to be retrieved from such tables, the entire table needs to be scanned
- This slows down the query retrieval process. To speed up query retrieval, indexes need to be created.
- When an index is created on a table, the index creates an order for the data rows or records in the table as shown in the following figure



Indexes

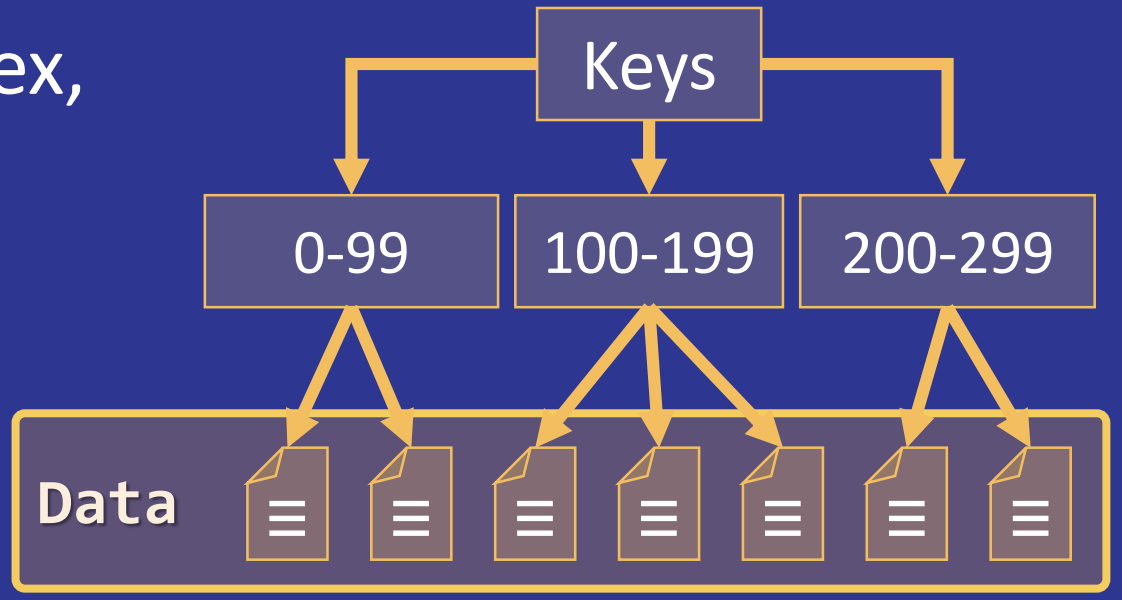
- Index speed up searching of values in a certain column or group of columns.
  - Usually implemented as B-trees.
- Indices can be built-in the table (clustered) or stored externally (non-clustered).
- Adding and deleting records in indexed tables is slower!
  - Indices should be used for big tables only (e.g. `50 000` rows).



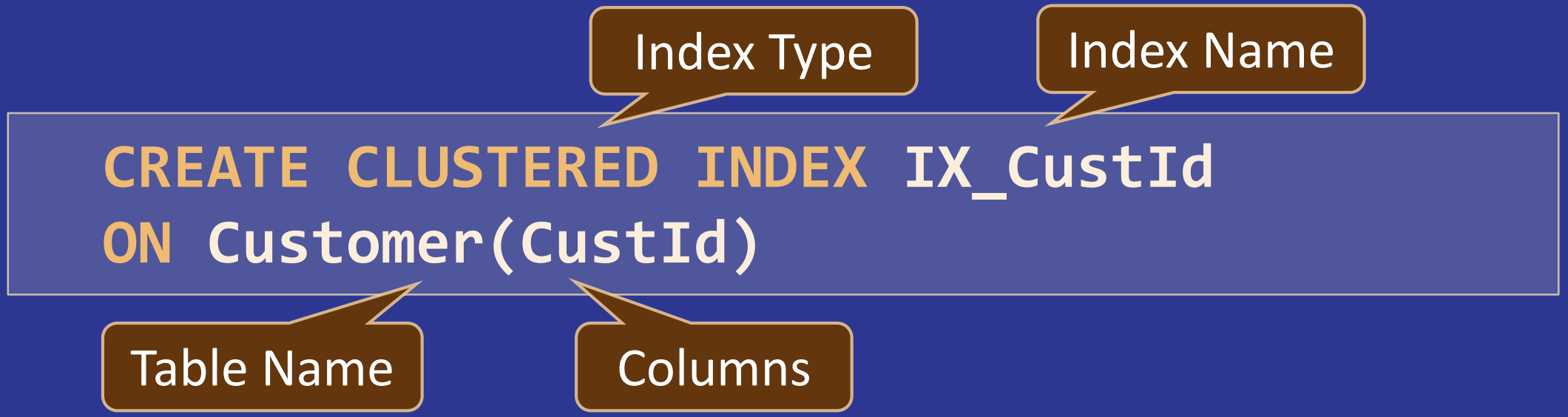


Clustered Indexes

- **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).



Index Syntax

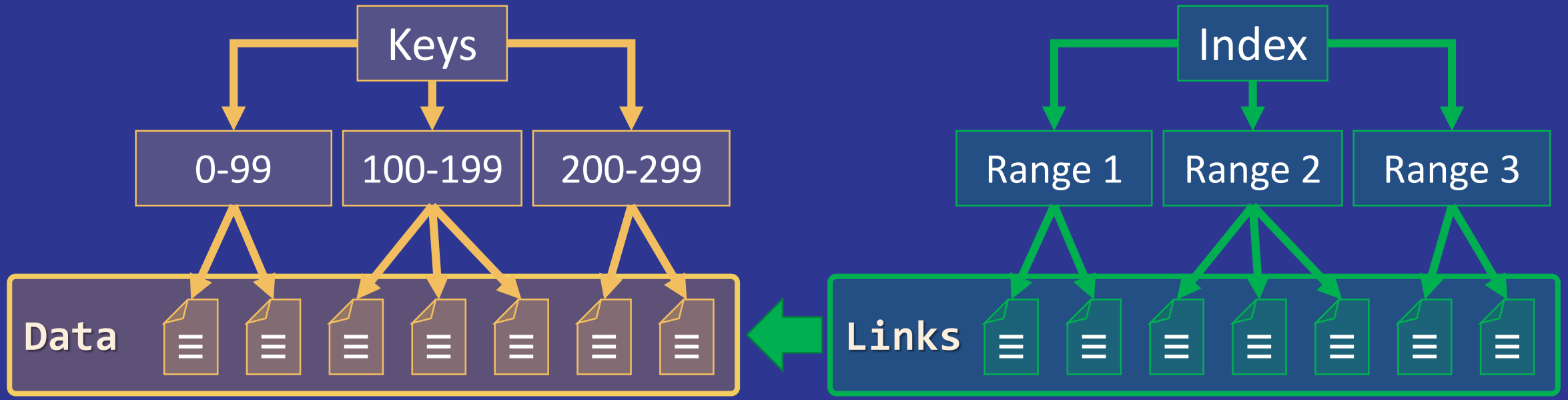


Non-Clustered Indexes (1)

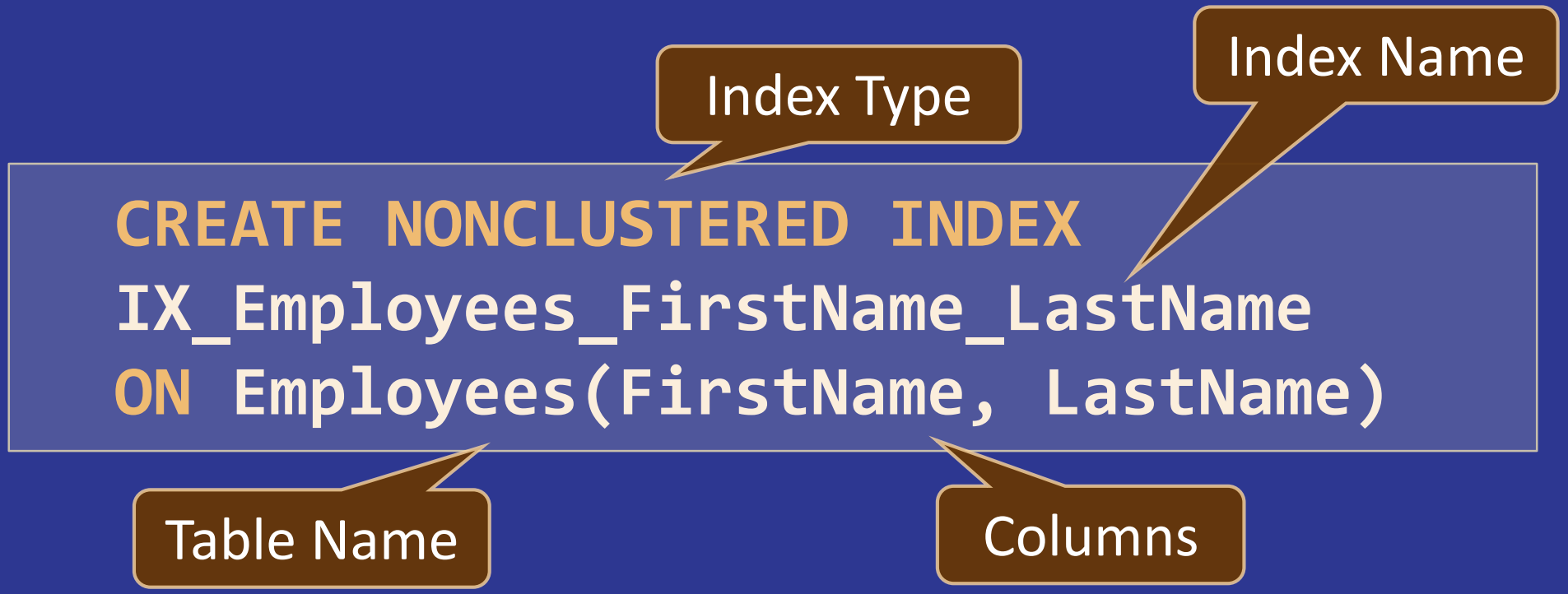
- Useful for fast retrieving a single record or a range of records
- Maintained in a separate structure in the DB
- Tend to be much narrower than the base table
  - Can locate the exact record(s) with less I/O
- Has at least one more intermediate level than the clustered index
  - Much less valuable if 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).



Index Syntax



Summary

1. Joins

```
SELECT * FROM Employees AS e
  JOIN Departments AS d ON
d.DepartmentId = e.DepartmentID
```

2. Subqueries are used to nest queries.
3. CTE's improve code reuse and readability.
4. Indices improve SQL search performance if used properly.