Joins, Subqueries and Indices

Data Retrieval and Performance

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Questions

sli.do

#java-db

Table of Contents

JOINs

Gathering Data from Multiple Tables

Data from Multiple Tables

- Sometimes you need data from several tables:

Employees

employee_name	department_id
Edward	3
John	NULL

Departments

department_id	department_name
3	Sales
4	Marketing
5	Purchasing

employee_name	department_id	department_name
Edward	3	Sales

Cartesian Product

- This will produce **Cartesian product**:

```
SELECT last_name, name AS department_name
FROM employees, departments;
```

- The result:

last_name	department_name
Gilbert	Engineering
Brown	Engineering
…	…
Gilbert	Sales
Brown	Sales

Cartesian Product

- Each row in the first table is paired with **all** the rows in the second table
  - When there is **no relationship** defined between the two tables
- Formed when:
  - A join condition is omitted
  - A join condition is invalid
- To avoid, always include a valid **JOIN condition**

JOINS

- **JOINS** – used to collect data from **two** or **more** tables
- Types:

INNER JOIN

LEFT JOIN

RIGHT JOIN

OUTER (UNION) JOIN

CROSS JOIN

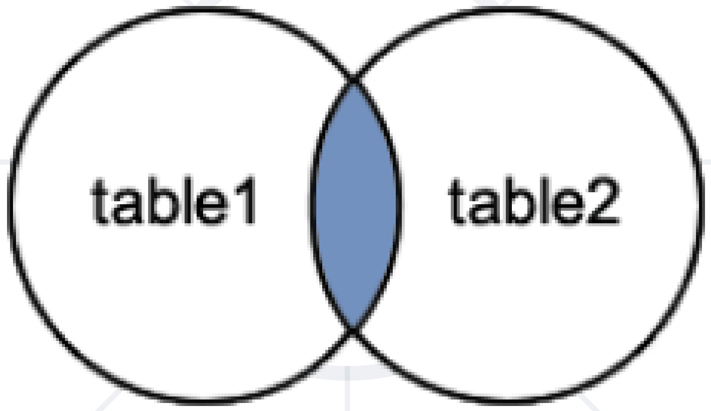
Tables

id	name	course_id
1	Alice	1
2	Michael	1
3	Caroline	2
4	David	5
5	Emma	NULL

id	name
1	HTML5
2	CSS3
3	JavaScript
4	PHP
5	MySQL

INNER JOIN

- Produces a set of records which **match in both tables**



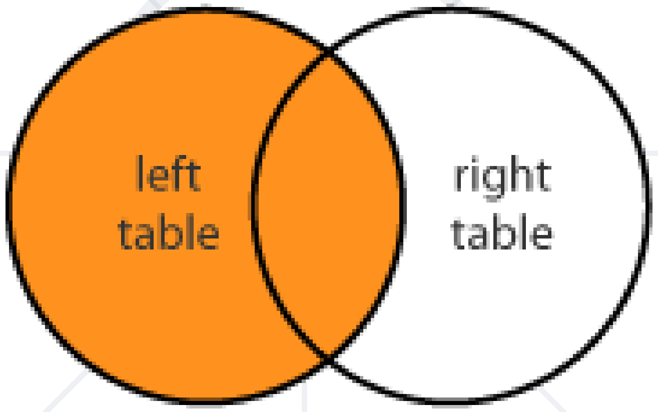
```
SELECT students.name, courses.name
FROM students
INNER JOIN courses #or just JOIN
ON students.course_id = courses.id
```

Join Conditions

students_name	courses_name
Alice	HTML5
Michael	HTML5
Caroline	CSS3
David	MySQL

LEFT JOIN

- Matches every entry in **left** table regardless of match in the **right**



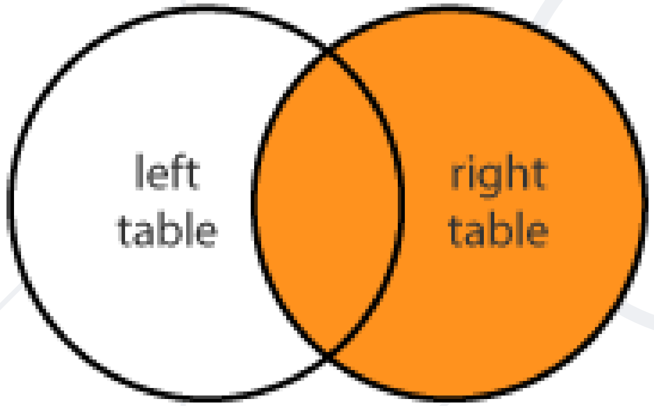
```
SELECT students.name, courses.name
FROM students
LEFT JOIN courses
ON students.course_id = courses.id
```

Join Conditions

students_name	courses_name
Alice	HTML5
Michael	HTML5
Caroline	CSS3
David	MySQL
Emma	NULL

RIGHT JOIN

- Matches every entry in **right** table regardless of match in the **left**



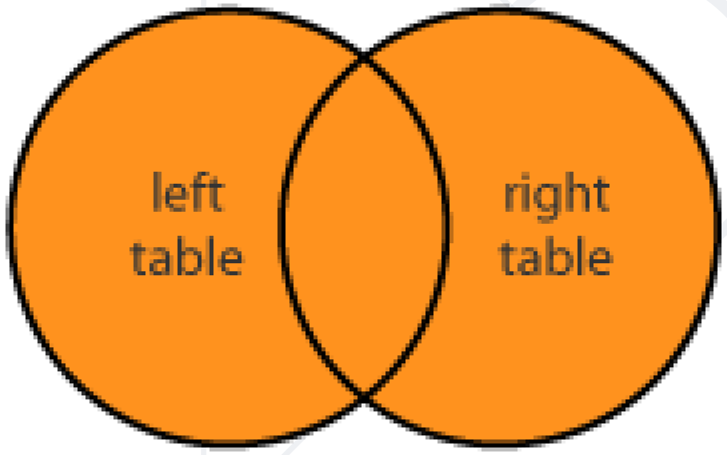
```
SELECT students.name, courses.name
FROM students
RIGHT JOIN courses
ON students.course_id = courses.id
```

Join Conditions

students_name	courses_name
Alice	HTML5
Michael	HTML5
Caroline	CSS3
NULL	JavaScript
NULL	PHP
David	MySQL

OUTER (FULL JOIN)

- Returns all records in both tables regardless of **any** match
  - Less useful than **`INNER`**, **`LEFT`** or **`RIGHT JOINs`** and it's **not implemented in MySQL**
  - We can use **`UNION`** of a **`LEFT`** and **`RIGHT JOIN`**



UNION of LEFT and RIGHT JOIN

```
SELECT students.name, courses.name
FROM students
LEFT JOIN courses
ON students.course_id = courses.id

UNION

SELECT students.name, courses.name
FROM students
RIGHT JOIN courses
ON students.course_id = courses.id
```

students_name	courses_name
Alice	HTML5
Michael	HTML5
Caroline	CSS3
David	MySQL
Emma	NULL
NULL	JavaScript
NULL	PHP

Cross Join

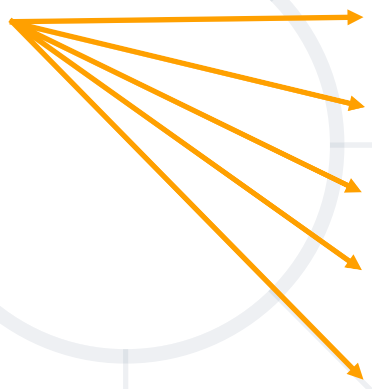
- Produces a set of associated rows of two tables
  - Multiplication of each row in the first table with each in second
  - The result is a **Cartesian** product, when there's **no condition** in the **WHERE** clause

```
SELECT * FROM courses AS c
 CROSS JOIN students AS s;
```

No Join Conditions

Cross Join

id	name
1	HTML5
2	CSS3
3	JavaScript
4	PHP
5	MySQL



id	name	course_id
1	Alice	1
2	Michael	1
3	Caroline	2
4	David	5
5	Emma	NULL

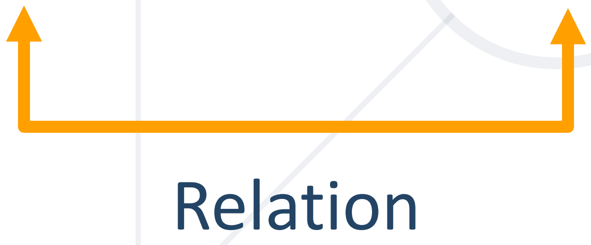
Result

course_id	course_name	student_id	student_name
1	HTML5	1	Alice
1	HTML5	2	Michael
1	HTML5	3	Caroline
…	…	…	…

Join Overview

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

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



Join Overview: INNER JOIN

employee_name	department_id
Sally	13
John	10
Michael	12
Bob	22
Robin	7
Jessica	8

department_id	department_name
9	Executive
8	Sales
11	Marketing
12	HR
18	Accounting
22	Engineering

employee_name	department_id	department_name
Michael	12	HR
Bob	22	Engineering
Jessica	8	Sales

Join Overview: LEFT JOIN

employee_name	department_id
Sally	13
Jessica	8
Michael	22
Bob	11

department_id	department_name
8	Sales
12	HR
18	Accounting
22	Engineering

employee_name	department_id	department_name
Sally	13	NULL
Jessica	8	Sales
Michael	22	Engineering
Bob	11	NULL

Join Overview: RIGHT JOIN

employee_name	department_id
Sally	13
Jessica	8
Michael	22
Bob	11

department_id	department_name
8	Sales
12	HR
18	Accounting
22	Engineering

employee_name	department_id	department_name
Jessica	8	Sales
NULL	12	HR
NULL	18	Accounting
Michael	22	Engineering

Problem: Managers

- Get information about the **first 5 managers** in the "soft_uni" database
  - **`id`**
  - **`full_name`**
  - **`department_id`**
  - **`department_name`**

employee_id	full_name	department_id	name
3	Roberto Tamburello	10	Finance
4	Rob Walters	2	Tool Design
6	David Bradley	5	Purchasing
12	Terri Duffy	1	Engineering
21	Peter Krebs	8	Production Control

Solution: Managers

```
SELECT e.employee_id, CONCAT(first_name, ' ',
last_name) AS 'full_name', d.department_id,
d.name
FROM employees AS e
RIGHT JOIN departments AS d
ON d.manager_id = e.employee_id
ORDER BY e.employee_id LIMIT 5;
```

Subqueries

Query Manipulation On Multiple Levels

Subqueries

- Subqueries – SQL query inside a larger one
- Can be nested in **SELECT**, **INSERT**, **UPDATE**, **DELETE**
  - Usually added within a **WHERE** clause

```
SELECT * FROM students
WHERE course_id = 1;
```

id	name	course_id
1	Alice	1
2	Michael	1

Subquery

Problem: Higher Salary

- **Count** the number of employees who receive salary, **higher** than the average
  - Use "soft_uni" database

employee_id	first_name	last_name	…
216	Mike	Seamans	…
178	Barbara	Moreland	…
…	…	…	…

Table "employees"

count
88

Solution: Higher Salary

```
SELECT COUNT(e.employee_id) AS 'count'
FROM employees AS e
WHERE e.salary >
(
    SELECT AVG(salary) AS
    'average_salary' FROM employees
);
```

Indices

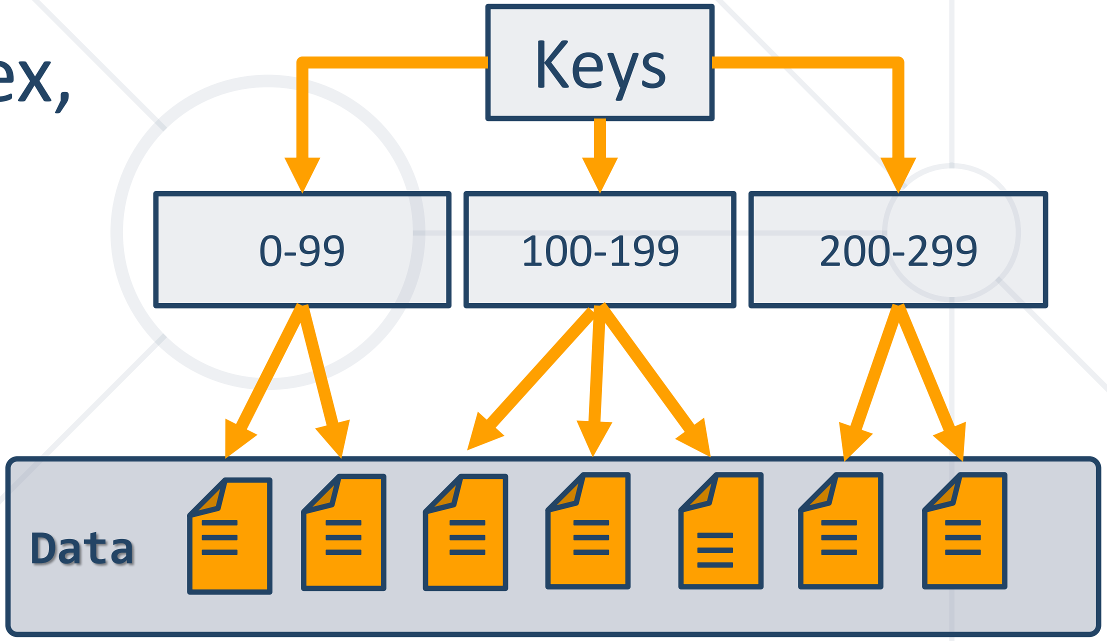
Clustered and Non-Clustered Indices

Indices

- Structures associated with a table or view that speeds retrieval of rows
  - 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)

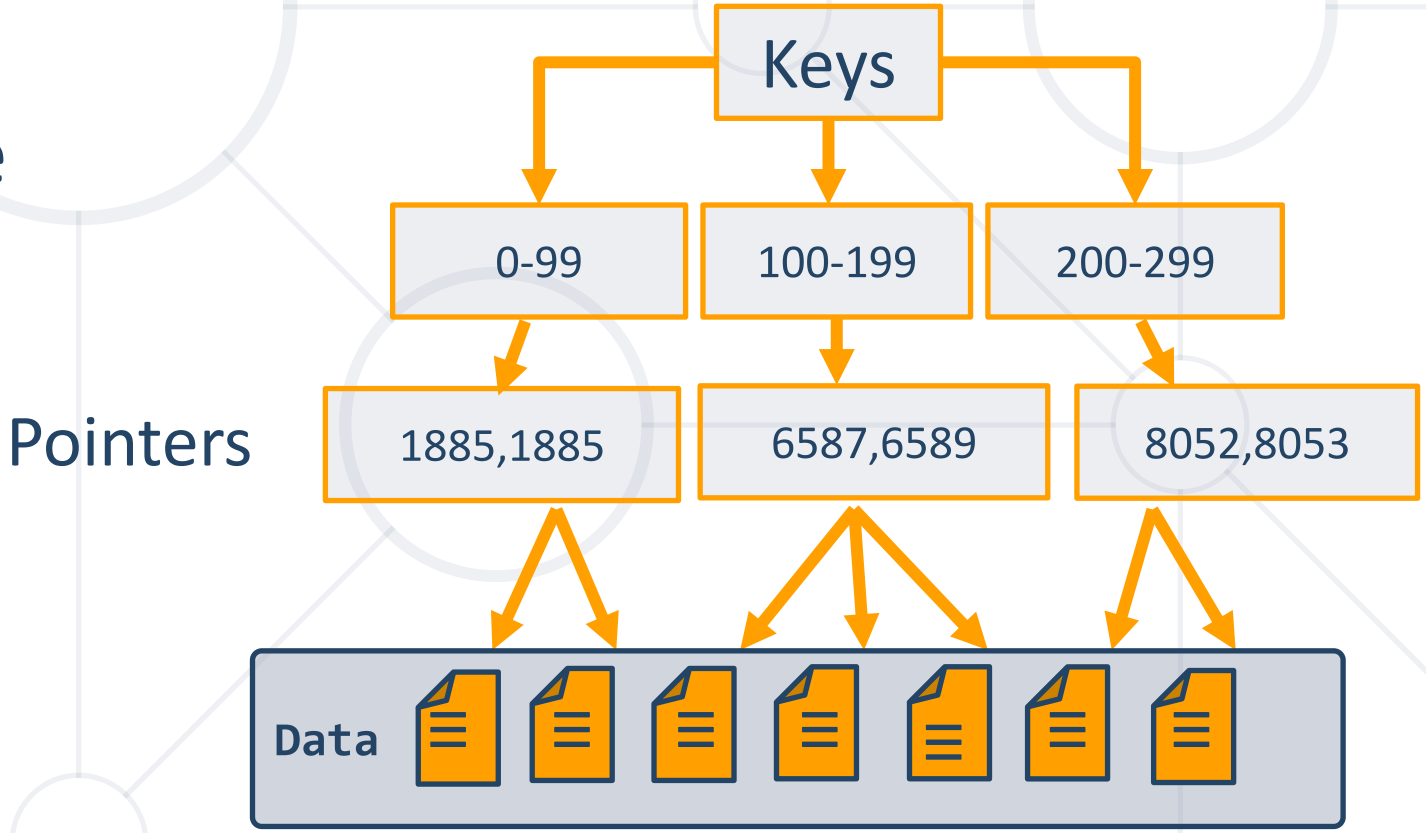
Clustered Indices

- **Clustered** index **determine the order** of data
  - 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)



Non-Clustered Indices

- Useful for fast retrieving a **single record** or a **range** of records
  - Each **key value entry** has a pointer to the data row that contains the key value
- Maintained in a separate
- Structure in the DB



Indices Syntax

```
CREATE INDEX
  ix_users_first_name_last_name
ON users(first_name, last_name);
```

Table Name

Columns

Summary

- Joins

```
SELECT * FROM employees AS e
  JOIN departments AS d ON
d.department_id = e.department_id
```

- **Subqueries** are used to nest queries
- Indices improve SQL search **performance** if used properly

Questions?

Software University

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

Finance Academy

SoftUni Diamond Partners

Software University

SmartIT

createX

VIVACOM

INDEAVR
Serving the high achievers

Coca-Cola HBC
Bulgaria

Postbank
Решения за твоето утре

AMBITIONED

SUPER
HOSTING
.BG

DRAFT
KINGS
THE CROWN IS YOURS

PHAR
VISION

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

License