**MYSQL**

**&**

**NORMALIZATION**



JOINS AND SUBQUERY – CHRISHANTH

NORMALIZATION - NISHARUGAAN

Table of Contents

[MY SQL JOIN QUERY 4](#_Toc103783369)

[What is a join Query? 4](#_Toc103783370)

[TYPES OF JOINS 4](#_Toc103783371)

[INNER JOIN 4](#_Toc103783372)

[LEFT JOIN 4](#_Toc103783373)

[RIGHT JOIN 5](#_Toc103783374)

[CROSS JOIN 5](#_Toc103783375)

[INNER JOIN 6](#_Toc103783376)

[INNER JOIN SYNTAX 6](#_Toc103783377)

[EXAMPLE 7](#_Toc103783378)

[LEFT JOIN 8](#_Toc103783379)

[LEFT JOIN SYNTAX 8](#_Toc103783380)

[EXAMPLE 9](#_Toc103783381)

[RIGHT JOIN 10](#_Toc103783382)

[RIGHT JOIN SYNTAX 10](#_Toc103783383)

[EXAMPLE 11](#_Toc103783384)

[CROSS JOIN 12](#_Toc103783385)

[CROSS JOIN SYNTAX 12](#_Toc103783386)

[EXAMPLE 13](#_Toc103783387)

[SELF JOIN 14](#_Toc103783388)

[SELF JOIN SYNTAX 14](#_Toc103783389)

[EXAMPLE 14](#_Toc103783390)

[MYSQL SUB QUERIES 15](#_Toc103783391)

[What are SUB QUERIES? 15](#_Toc103783392)

[What is the use of SUB QUERIES? 15](#_Toc103783393)

[The SQL IN Operators 15](#_Toc103783394)

[IN SYNTAX 15](#_Toc103783395)

[EXAMPLE 15](#_Toc103783396)

[The SQL ANY and ALL Operators 16](#_Toc103783397)

[SQL ANY Operator 16](#_Toc103783398)

[ANY SYNTAX 16](#_Toc103783399)

[SQL ALL Operator 16](#_Toc103783400)

[ANY SYNTAX 16](#_Toc103783401)

[DBMS Normalization 17](#_Toc103783402)

[What is Normalization? 17](#_Toc103783403)

[Purpose of Normalization? 17](#_Toc103783404)

[Normal Forms in SQL 17](#_Toc103783405)

[1NF Normalization 18](#_Toc103783406)

[**Example** 18](#_Toc103783407)

[2NF Normalization 19](#_Toc103783408)

[**Example** 19](#_Toc103783409)

[3NF Normalization 20](#_Toc103783410)

# MY SQL JOIN QUERY

## What is a join Query?

Gathering details from two or more tables by combining it rows.

# TYPES OF JOINS

## INNER JOIN

## LEFT JOIN



## RIGHT JOIN



## CROSS JOIN



# INNER JOIN



**INNER JOIN** returns matching records from the table.

## INNER JOIN SYNTAX

**SELECT** column\_name(s)

**FROM** table1

**INNER JOIN** table2

**ON** table1.column\_name = table2.column-name**;**

Football Table

|  |  |  |
| --- | --- | --- |
| ID | NAME | CITY |
| 1 | Tom | Colombo |
| 5 | Holland | Trinco |
| 8 | Peter | Jaffna |

Basketball Table

|  |  |  |
| --- | --- | --- |
| ID | NAME | CITY |
| 2 | Parker | Kandy |
| 8 | Peter | Jaffna |
| 5 | Holland | Trinco |

## EXAMPLE

**SELECT** \*

**FROM** Football

**INNER JOIN** Basketball

**ON** Football.Id = Basketball.Id**;**

Results

|  |  |  |
| --- | --- | --- |
| ID | NAME | CITY |
| 8 | Peter | Jaffna |
| 5 | Holland | Trinco |

**NOTE** :- We can also able to achieve this results using **WHERE** clause too.

**SELECT** \*

**FROM** Football, Basketball

**WHERE** Football.Id = Basketball.Id**;**

# LEFT JOIN



**LEFT JOIN** return all the records in left table and matching records in right table.

## LEFT JOIN SYNTAX

**SELECT** column\_name(s)

**FROM** table1

**LEFT JOIN** table2

**ON** table1.column\_name = table2.column-name**;**

Football Table

|  |  |  |
| --- | --- | --- |
| ID | NAME | CITY |
| 1 | Tom | Colombo |
| 5 | Holland | Trinco |
| 8 | Peter | Jaffna |

Basketball Table

|  |  |  |
| --- | --- | --- |
| ID | NAME | CITY |
| 2 | Parker | Kandy |
| 8 | Peter | Jaffna |
| 5 | Holland | Trinco |

## EXAMPLE

**SELECT** \*

**FROM** Football

**LEFT JOIN** Basketball

**ON** Football.Id = Basketball.Id**;**

Results

|  |  |  |
| --- | --- | --- |
| ID | NAME | CITY |
| 1 | Tom | Colombo |
| 8 | Peter | Jaffna |
| 5 | Holland | Trinco |

# RIGHT JOIN



**RIGHT JOIN** return all the records in right table and matching records in left table.

## RIGHT JOIN SYNTAX

**SELECT** column\_name(s)

**FROM** table1

**RIGHT JOIN** table2

**ON** table1.column\_name = table2.column-name**;**

Football Table

|  |  |  |
| --- | --- | --- |
| ID | NAME | CITY |
| 1 | Tom | Colombo |
| 5 | Holland | Trinco |
| 8 | Peter | Jaffna |

Basketball Table

|  |  |  |
| --- | --- | --- |
| ID | NAME | CITY |
| 2 | Parker | Kandy |
| 8 | Peter | Jaffna |
| 5 | Holland | Trinco |

## EXAMPLE

**SELECT** \*

**FROM** Football

**RIGHT JOIN** Basketball

**ON** Football.Id = Basketball.Id**;**

Results

|  |  |  |
| --- | --- | --- |
| ID | NAME | CITY |
| 2 | Parker | Kandy |
| 8 | Peter | Jaffna |
| 5 | Holland | Trinco |

# CROSS JOIN



**CROSS JOIN** return all the records from both tables.

## CROSS JOIN SYNTAX

**SELECT** column\_name(s)

**FROM** table1

**CROSS JOIN** table2**;**

Team Table

|  |  |  |
| --- | --- | --- |
| ID | NAME | CITY |
| 1 | Tom | Colombo |
| 2 | Holland | Trinco |
| 3 | Peter | Jaffna |

Coach Table

|  |  |  |
| --- | --- | --- |
| ID | NAME | CITY |
| 4 | Parker | Kandy |
| 5 | Jonny | Jaffna |
| 6 | Depp | Trinco |

## EXAMPLE

**SELECT** \*

**FROM** Football

**CROSS JOIN** Basketball**;**

Results

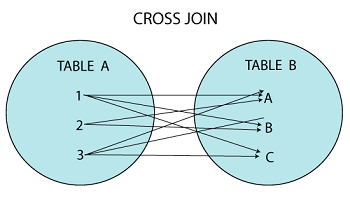
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | NAME | CITY | ID | NAME | CITY |
| 1 | Tom | Colombo | 4 | Parker | Kandy |
| 1 | Tom | Colombo | 5 | Jonny | Jaffna |
| 1 | Tom | Colombo | 6 | Depp | Trinco |
| 2 | Holland | Trinco | 4 | Parker | Kandy |
| 2 | Holland | Trinco | 5 | Jonny | Jaffna |
| 2 | Holland | Trinco | 6 | Depp | Trinco |
| 3 | Peter | Jaffna | 4 | Parker | Kandy |
| 3 | Peter | Jaffna | 5 | Jonny | Jaffna |
| 3 | Peter | Jaffna | 6 | Depp | Trinco |

* The results of this **CROSS JOIN** you see above has nine rows that’s by the Team Table has three rows and Coach Table has Three rows so

3×3=9 rows.

* Every single row of a table combines with next rows of the table.

The below image graphically represent the combinations of **CROSS JOIN**



# SELF JOIN

A **SELF JOIN** is a regular join, but the table is joined with itself.

## SELF JOIN SYNTAX

**SELECT** column\_name(s)

**FROM** table1 T1, table2 T2

**WHERE** condition**;**

Student Table

|  |  |  |  |
| --- | --- | --- | --- |
| STUDENT\_ID | NAME | COURSE\_ID | DURATION |
| 1 | Tom | 1 | 3 |
| 2 | Holland | 2 | 4 |
| 1 | Tom | 2 | 4 |
| 3 | Jonny | 3 | 2 |
| 2 | Holland | 3 | 5 |

## EXAMPLE

**SELECT** s1.STUDENT\_ID, s1.NAME

**FROM** Student AS s1, Student s2

**WHERE** s1.student-id = s2.student\_id

**AND** s1.course\_id <> s2.course\_id**;**

Results

|  |  |
| --- | --- |
| ID | NAME |
| 1 | Tom |
| 2 | Holland |
| 1 | Tom |
| 2 | Holland |

# MYSQL SUB QUERIES

## What are SUB QUERIES?

A MySQL subquery is a query nested within another query such as **SELECT , INSERT , UPDATE or DELETE.**

## What is the use of SUB QUERIES?

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

## The SQL IN Operators

The in operator allow u to make a query in a main query. Also we can use functions in the subquery such as **COUNT(), MAX(), MIN(), AVG()** ect..

## IN SYNTAX

**SELECT** Column\_name(s)

**FROM** table\_name

**WHERE** column\_name **IN**

(SELECT column\_name

FROM table\_name

WHERE condition);

## EXAMPLE

**SELECT** customerNumber, checkNumber, amount

**FROM** payments

**WHERE** amount **IN** (SELECT MAX(amount) FROM payments WHERE amount > 50);

## The SQL ANY and ALL Operators

The ANY and ALL operators allow you to perform a comparison between a single column value and a range of other values.

## SQL ANY Operator

Return a value if ANY of the sub query value meet the condition.

## ANY SYNTAX

**SELECT** Column\_name(s)

**FROM** table\_name

**WHERE** column\_name operator **ANY**

(SELECT column\_name

FROM table\_name

WHERE condition)**;**

## SQL ALL Operator

Returns the values if all of the sub queries value meets the condition.

Can able to use with **SELECT**, **WHERE** and **HAVING** statements.

## ANY SYNTAX

**SELECT** Column\_name(s)

**FROM** table\_name

**WHERE** column\_name operator **ALL**

(SELECT column\_name

FROM table\_name

WHERE condition)**;**

# DBMS Normalization

## **What is Normalization?**

* **Normalization is a database design technique that reduces data redundancy and eliminates unwanted characteristics.**
* **Normalization rules divides larger table into smaller tables and links them using relationships.**

## **Purpose of Normalization?**

* **The purpose of Normalization in SQL is to eliminate redundant data and ensure data is stored logically.**
* **Edgar Codd inventor of Relational Model proposed the theory of normalization with 1NF.**

## **Normal Forms in SQL**

* **1NF**
* **2NF**
* **3NF**

3NF

2NF

1NF

## **1NF Normalization**

* **In 1NF a table’s attribute would not be able to hold various values it will only be able to hold an attribute of single Value.**
* **Each record needs to be unique.**

### **Example**

|  |  |  |
| --- | --- | --- |
| **Stu\_No** | **Name** | **Courses** |
| **11** | **Sourav** | **Web, Android** |
| **12** | **Shiran** | **C++** |
| **13** | **Kishon** | **C++, Java** |

**There are some multiple values in courses column. We use 1NF method to resolve it as follows.**

|  |  |  |
| --- | --- | --- |
| **Stu\_No** | **Name** | **Courses** |
| **11** | **Sourav** | **Web** |
| **11** | **Sourav** | **Android** |
| **12** | **Shiran** | **C++** |
| **13** | **Kishon** | **C++** |
| **13** | **Kishon** | **Java** |

**There are some values getting repeated but there is just one value in every column.**

## **2NF Normalization**

* a relation is said to be in 2NF when it exists in 1NF, while the relation’s every non-prime attribute depends on every candidate key as a whole.
* If a relation is in 1NF and all the attributes of the non-primary keys are fully dependent on primary keys, then this relation is known to be in the 2NF or the Second Normal Form.

### **Example**

**Lecturer\_Table**

|  |  |  |
| --- | --- | --- |
| **Lecturer\_ID** | **Course** | **Lecturer\_Age** |
| **1001** | **Java** | **34** |
| **1001** | **C++** | **34** |
| **1204** | **Web** | **29** |
| **1212** | **Android** | **32** |
| **1212** | **Python** | **32** |

**Lecturer\_Detail\_Table**

|  |  |
| --- | --- |
| **Lecturer\_ID** | **Lecturer\_Age** |
| **1001** | **34** |
| **1204** | **29** |
| **1212** | **32** |

**Lecturer\_Course\_Table**

|  |  |
| --- | --- |
| **Lecturer\_ID** | **Course** |
| **1001** | **Java** |
| **1001** | **C++** |
| **1204** | **Web** |
| **1212** | **Android** |
| **1212** | **Python** |

## **3NF Normalization**

* In a relation that is in 1NF or 2NF, when none of the non-primary key attributes transitively depend on their primary keys, then we can say that the relation is in the third normal form of 3NF.
* 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.

Student\_Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Stu\_No** | **Name** | **Postcode** | **City** | **Province** |
| **11** | **Sourav** | **40000** | **Jaffna** | **North** |
| **12** | **Shiran** | **31000** | **Trincomalee** | **East** |
| **13** | **Kishon** | **90000** | **Badulla** | **Uva** |
| **14** | **Stephan** | **00800** | **Borella** | **West** |
| **15** | **Biet** | **20400** | **Peradeniya** | **Central** |

**Student\_Table**

|  |  |  |
| --- | --- | --- |
| **Stu\_No** | **Name** | **Postcode** |
| **11** | **Sourav** | **40000** |
| **12** | **Shiran** | **31000** |
| **13** | **Kishon** | **90000** |
| **14** | **Stephan** | **00800** |
| **15** | **Biet** | **20400** |

**Student\_City\_Table**

|  |  |  |
| --- | --- | --- |
| **Postcode** | **City** | **Province** |
| **40000** | **Jaffna** | **North** |
| **31000** | **Trincomalee** | **East** |
| **90000** | **Badulla** | **Uva** |
| **00800** | **Borella** | **West** |
| **20400** | **Peradeniya** | **Central** |

THANKYOU