PRIMARY KEY — Definition & Rules (Easy Explanation)

□ What is a Primary Key?

A **Primary Key** is a column (or a group of columns) in a table that **uniquely identifies each record**.

□ It ensures that **no two rows are the same**.

□ Simple Example

Parent Table (Primary Key)

```
CREATE TABLE Departments (
    dept_id INT PRIMARY KEY,
    dept_name VARCHAR(50)

);
```

Here, `dept_id` uniquely identifies each department.

□ Rules of Primary Key

1□ Uniqueness Rule

- Each primary key value must be **unique**
- No duplicates allowed

□ Not allowed:

FOREIGN KEY (FK)

□ What is a Foreign Key?

A **Foreign Key** is a column in one table that **references the primary key of another table**.

□ It creates a **relationship** between tables.

□ Why Use Foreign Keys?

✔ Maintains **data integrity**
✔ Prevents invalid data
✔ Connects related tables

Example:

- A student must belong to an **existing department**
- You cannot assign a department that doesn't exist

□ Example: Foreign Key

Parent Table (Primary Key)

```
CREATE TABLE Departments (
    dept_id INT PRIMARY KEY,
    dept_name VARCHAR(50)
);
```

Child Table (Foreign Key)

```
CREATE TABLE Students (
    student_id INT PRIMARY KEY,
    name VARCHAR(50),
    dept_id INT,
    FOREIGN KEY (dept_id) REFERENCES Departments(dept_id)
);
```

Here:

- `dept_id` in `Students` is a **foreign key**
- It refers to `dept_id` in `Departments`

□ Inserting Valid Data

```
INSERT INTO Departments VALUES (1, 'Computer Science');

INSERT INTO Students VALUES (101, 'Ali', 1);
```

✔ Works because department `1` exists

□ Inserting Invalid Data

```
INSERT INTO Students VALUES (102, 'Sara', 5);
```

□ Error — department `5` does not exist

Here is a **complete and clear example** of **Department** and **Students** tables **with data**, showing **Primary Key** and **Foreign Key** relationship.

□ 1□ Department Table (Parent Table)

Table Creation

```
CREATE TABLE Department (
    dept_id INT PRIMARY KEY,
    dept_name VARCHAR(50)
);
```

Sample Data (Insert Records)

```
INSERT INTO Department (dept_id, dept_name) VALUES
(1, 'Computer Science'),
(2, 'Mathematics'),
(3, 'Physics');
```

Department Table Data

dept_id	dept_name
1	Computer Science
2	Mathematics
3	Physics

✔ `dept_id` is **PRIMARY KEY**
✔ Each department has a **unique ID**

□□ 2□ Students Table (Child Table)

Table Creation

```
CREATE TABLE Students (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(50),
    dept_id INT,
    FOREIGN KEY (dept_id) REFERENCES Department(dept_id)
);
```

Sample Data (Insert Records):-

```
INSERT INTO Students (student_id, student_name, dept_id) VALUES
(101, 'Ali', 1),
(102, 'Sara', 1),
(103, 'Rahul', 2),
(104, 'Neha', 3);
```

Students Table Data

student_id	student_name	dept_id
101	Ali	1
102	Sara	1
103	Rahul	2
104	Neha	3

✔ `student_id` is **PRIMARY KEY**
✔ `dept_id` is **FOREIGN KEY**
✔ Each student belongs to a valid department

□ Relationship Explanation

- `Department.dept_id` → **Primary Key**
- `Students.dept_id` → **Foreign Key**
- One department can have **many students**
- One student belongs to **one department**

This is a **One-to-Many relationship**.

Method 1:-

Query USING SUBQUERY (Most Important)

```
SELECT student_id, student_name
FROM Students

WHERE dept_id = (

    SELECT dept_id
    FROM Department
    WHERE dept_name = 'Computer Science'
);
```

□ Explanation

- Inner query gets `dept_id` of **Computer Science**
- Outer query finds students with that `dept_id`
- No JOIN used

□ Output Example (Both Queries)

student_id	student_name
101	Ali
102	Sara

✅ Query: Students with Department Name

```
SELECT s.student_id, s.student_name, d.dept_name
FROM Students s
JOIN Department d
ON s.dept_id = d.dept_id
WHERE d.dept_name = 'Computer Science';
```

🔍 Explanation

1. `FROM Students s JOIN Department d ON s.dept_id = d.dept_id`
   - Combines **Students** with **Department** based on `dept_id`.
2. `WHERE d.dept_name = 'Computer Science'`
   - Filters **only Computer Science students**.
3. `SELECT s.student_id, s.student_name, d.dept_name`
   - Shows **student details + department name**.

📊 Output Example

student_id	student_name	dept_name
101	Ali	Computer Science
102	Sara	Computer Science

🚀 Advantages of this JOIN

- Easy to **add more columns** (e.g., student age, department head)
- Can filter by **any department** easily
- Efficient for **large tables**

1️⃣ LEFT JOIN Version

```
SELECT s.student_id, s.student_name, d.dept_name
FROM Students s
LEFT JOIN Department d
ON s.dept_id = d.dept_id
WHERE d.dept_name = 'Computer Science';
```

🔍 Explanation

- `LEFT JOIN` includes **all students**, even if they don't have a matching department.
- `WHERE d.dept_name = 'Computer Science'` **still filters only Computer Science students**, so output is the **same as INNER JOIN** here.

✅ Output:

student_id	student_name	dept_name
101	Ali	Computer Science
102	Sara	Computer Science

Note: If you **remove the WHERE filter**, LEFT JOIN would show students **without a department** as `NULL`.

LEFT JOIN without WHERE

```
SELECT s.student_id, s.student_name, d.dept_name
FROM Students s
LEFT JOIN Department d
ON s.dept_id = d.dept_id;
```

student_id	student_name	dept_name
101	Ali	Computer Science
102	Sara	Computer Science
103	Rahul	Mathematics
104	Neha	Physics

---

2 RIGHT JOIN Version

```
SELECT s.student_id, s.student_name, d.dept_name
FROM Students s
RIGHT JOIN Department d
ON s.dept_id = d.dept_id
WHERE d.dept_name = 'Computer Science';
```

Explanation

- `RIGHT JOIN` includes **all departments**, even if they have no students.
- `WHERE d.dept_name = 'Computer Science'` filters only **Computer Science**, so output is **the same as INNER JOIN**.

Output:

student_id	student_name	dept_name
101	Ali	Computer Science
102	Sara	Computer Science

Key Notes

Join Type	What it includes	Notes in this example
INNER JOIN	Only matching students & departments	2 Computer Science students
LEFT JOIN	All students + matching departments	Same as INNER JOIN with WHERE
RIGHT JOIN	All departments + matching students	Same as INNER JOIN with WHERE

Let’s make a **visual diagram** showing **INNER JOIN, LEFT JOIN, and RIGHT JOIN** using your **Students** and **Department** tables. This makes it **super easy to remember for exams**.

□ Tables for Reference

Department Table

dept_id	dept_name
1	Computer Science
2	Mathematics
3	Physics

Students Table

student_id	student_name	dept_id
101	Ali	1
102	Sara	1
103	Rahul	2
104	Neha	3
105	Aman	NULL

□ Visual Diagram of JOINs

1□ INNER JOIN

- Shows **only matching rows** (students with a valid department)

student_id	student_name	dept_name
101	Ali	Computer Science
102	Sara	Computer Science
103	Rahul	Mathematics
104	Neha	Physics

□ Student 105 (Aman)**not included** because no department

---

2 LEFT JOIN

- Shows **all students**, even if they don't have a department
- Departments for missing students are `NULL`

student_id	student_name	dept_name
101	Ali	Computer Science
102	Sara	Computer Science
103	Rahul	Mathematics
104	Neha	Physics
105	Aman	NULL

 Includes Aman (no department)

---

3 RIGHT JOIN

- Shows **all departments**, even if no student belongs to them
- Students for empty departments are `NULL`

student_id	student_name	dept_name
101	Ali	Computer Science
102	Sara	Computer Science
103	Rahul	Mathematics
104	Neha	Physics
NULL	NULL	SomeDept

 Includes departments with no students (we don't have an empty department here, but if we added `dept_id = 4, dept_name='Chemistry'`, it would show `NULL` for student)

Quick Memory Trick

- **INNER JOIN** → Only **matches**
- **LEFT JOIN** → All **left table** (Students) + matches
- **RIGHT JOIN** → All **right table** (Department) + matches

 Think **LEFT table = Students**, **RIGHT table = Department** in your query.