GROUP MEMBER: NKURANGA BAZIGA CALEB 28845

ISHIMWE MUNEZA PRINCE 27816

IZERE SABIN PATIENCE 27969

**DATABASE DEVELOPMENT WITH PL/SQL GROUP WORK**

NO1: Create a table with appropriate columns and apply relevant constraints (e.g., primary key, foreign key, unique, not null).

Firstly before creating table we created database which we named it schooldb and then

We created three tables:

1. **Students** – contains details of students with a primary key and unique email constraint.
2. **Courses** – stores courses with a primary key and a check constraint for credit hours.
3. **Enrollments** – junction table linking students and courses with foreign keys.

**Constraints Applied:**

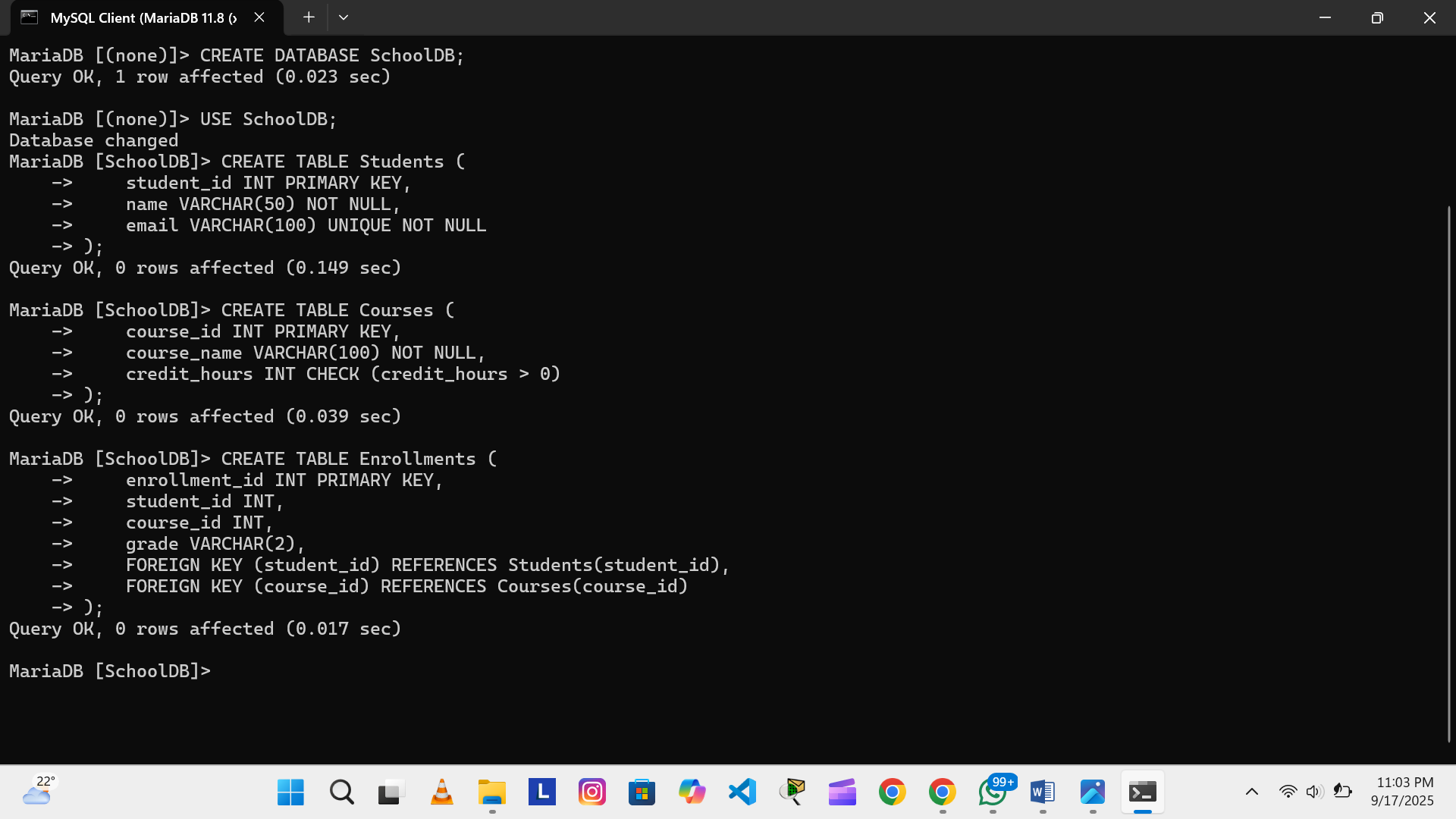
* PRIMARY KEY on all tables.
* UNIQUE on email in Students.
* CHECK to ensure credit hours > 0.
* FOREIGN KEY references between Students and Courses.

**Inserting data :  
 after creating table we inserted data so that the query of joining will work smoothly  
student table**

|  |  |  |
| --- | --- | --- |
| Student \_id | Names | Email |
| 1 | Alice | alice@gmail.com |
| 2 | Bob | bob@gmail.com |
| 3 | Charlie | [charlie@gmail.com](mailto:charlie@gmail.com) |
| 4 | diana | diana@gmail.com |

**Course table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Course\_id | Course\_name | | Credit\_hours | |
| 101 | Database systems | | 3 | |
| 102 | Computer networks | | 4 | |
| 103 | Operating systems | | 3 | |
| 104 | Artificial intelligence | | 4 | |
| Enrollment\_id | | Student\_id | | Course\_id | | Grade |
| 1 | | 1 | | 101 | | A |
| 2 | | 1 | | 102 | | B |
| 3 | | 2 | | 103 | | A |
| 4 | | 3 | | 101 | | C |



2. Perform different types of joins (INNER, LEFT, RIGHT, FULL) using your table(s).

With an **INNER JOIN**, the database looks at both tables and only keeps the rows where there is a perfect match on both sides — so only students linked to courses through enrollments appear.

With a **LEFT JOIN**, the database takes **all the rows from the left table (Students)** first, then tries to match them with enrollments and courses. If a student has no match, the database still keeps them but fills the missing course and grade with NULL.

**RIGHT JOIN**  
This join keeps **all courses** from the course list, even if nobody is enrolled in them. That’s why the Artificial Intelligence course still shows up in the results, but without any student attached to it. Students not linked to a course won’t appear here.

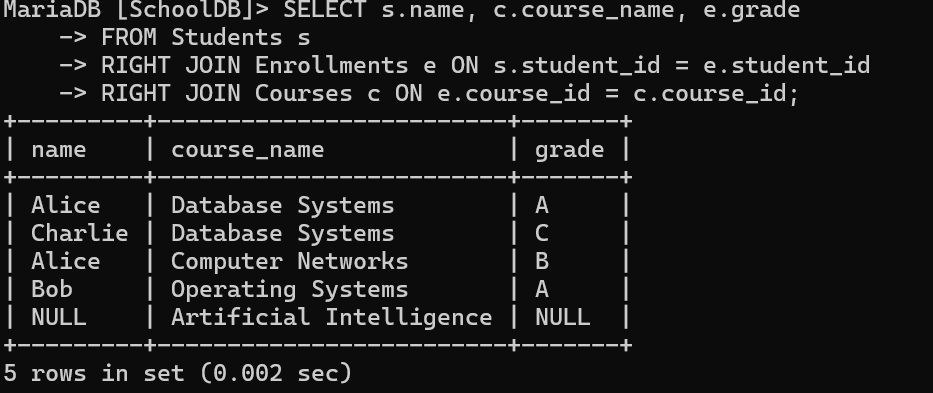
**FULL JOIN**  
This join combines both sides — it shows **all students and all courses**, whether they match or

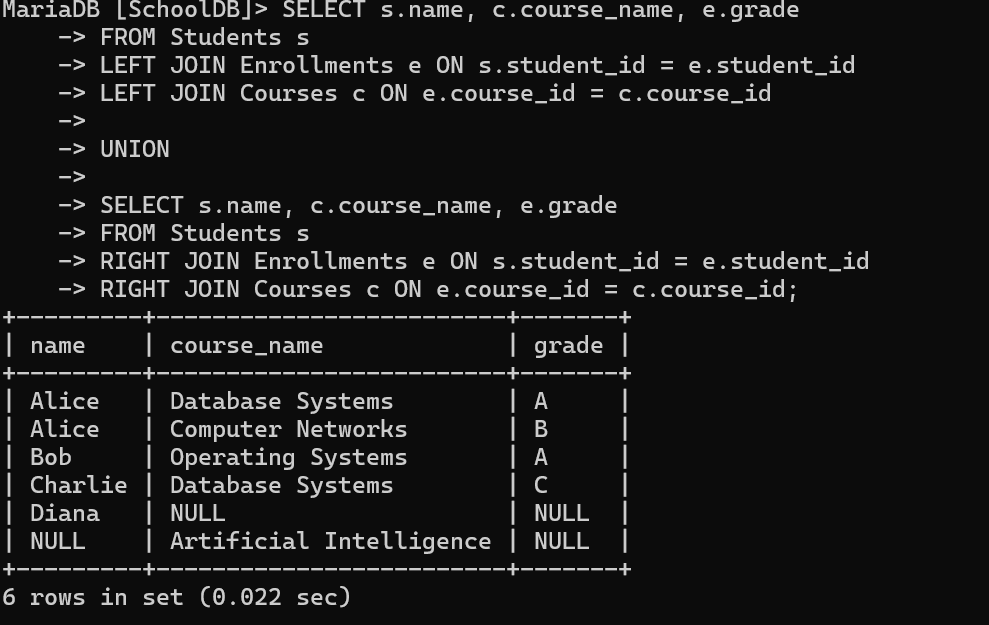
not. That means you’ll see Diana (who has no course) and the Artificial Intelligence course (which has no student) alongside the normal enrollments.

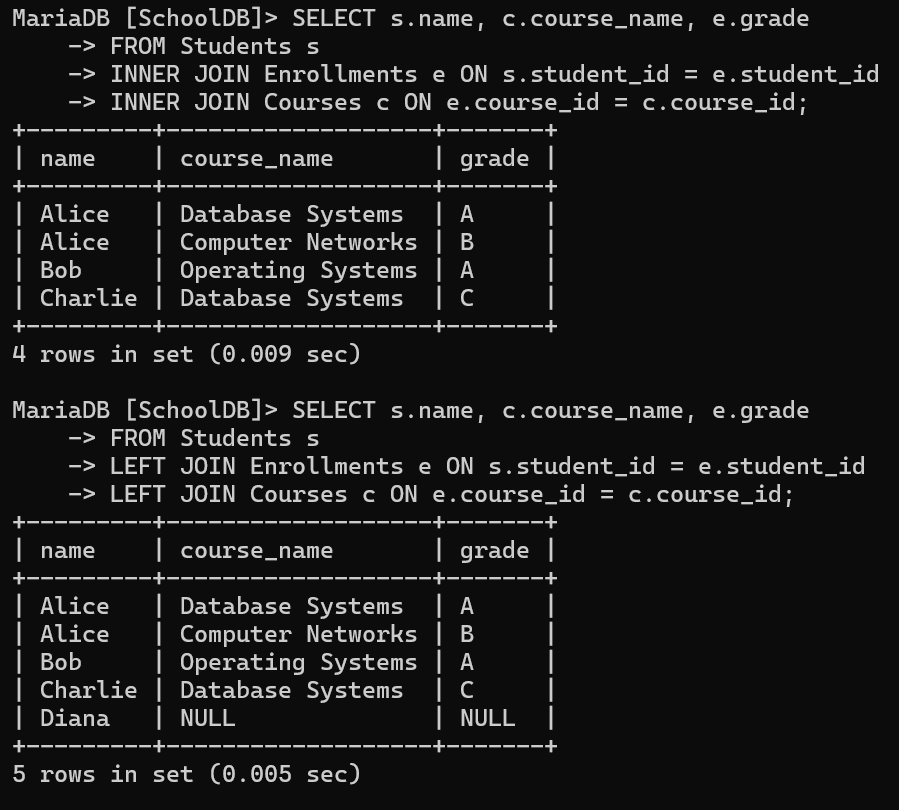
So to sum up all four in one line:

* **INNER JOIN** → Only matched student–course pairs.
* **LEFT JOIN** → All students, matched or not.
* **RIGHT JOIN** → All courses, matched or not.
* **FULL JOIN** → Everything from both sides, whether matched or not.

**RIGHT JOIN**

   
  
**Full join**



**INNERJOIN AND LEFT JOIN**  


3. Create an index to optimize query performance.  
 **Index:** It’s like a shortcut in a book. Creating an index on student\_id or course\_id helps the database **find data faster** when searching or joining tables.

4. Create a view to simplify data access.  
**View:** A view is like a **saved query**. Instead of writing a long join every time, you can create a view that shows **students, their courses, and grades** all together. Then you just ask the view for the information,

and it gives you the result.  
