**Table 1: Students**

| **StudentID** | **Name** | **CourseID** |
| --- | --- | --- |
| 1 | Alice Johnson | 201 |
| 2 | Bob Smith | 202 |
| 3 | Charlie Brown | 203 |
| 4 | David Williams | NULL |
| 5 | Emma Davis | 202 |

**Table 2: Courses**

| **CourseID** | **CourseName** |
| --- | --- |
| 201 | Mathematics |
| 202 | Physics |
| 203 | Chemistry |
| 204 | Computer Sci |

SQL Questions Based on the Above Tables

1. Find the names of students and their course names for those who are enrolled in a course.
2. List all students and the names of their courses. If a student is not enrolled in any course, show NULL for the course name.
3. List all courses and the names of students enrolled in those courses. If a course does not have any students, show NULL for the student name.
4. List all students and all courses. If a student is not enrolled in a course, show NULL for the course name, and if a course has no students enrolled, show NULL for the student name.
5. List the student names and course names, assuming both the Students and Courses tables have the same CourseID column.

**1. Find the names of students and their course names for those who are enrolled in a course.**

**Query:**

SELECT s.Name AS StudentName, c.CourseName

FROM Students s

INNER JOIN Courses c ON s.CourseID = c.CourseID;

**Explanation:**

* **INNER JOIN** retrieves only the records where there is a match between the Students and Courses tables based on the CourseID.
* This means only students who are enrolled in a course will be shown in the result.

**Result:**

| **StudentName** | **CourseName** |
| --- | --- |
| Alice Johnson | Mathematics |
| Bob Smith | Physics |
| Charlie Brown | Chemistry |
| Emma Davis | Physics |

**2. List all students and the names of their courses. If a student is not enrolled in any course, show NULL for the course name.**

**Query:**

SELECT s.Name AS StudentName, c.CourseName

FROM Students s

LEFT JOIN Courses c ON s.CourseID = c.CourseID;

**Explanation:**

* **LEFT JOIN** returns all records from the Students table, and matching records from the Courses table.
* If there is no matching CourseID, it will return NULL for the CourseName.
* This means every student is included in the result, even if they are not enrolled in any course (in this case, "David Williams").

**Result:**

| **StudentName** | **CourseName** |
| --- | --- |
| Alice Johnson | Mathematics |
| Bob Smith | Physics |
| Charlie Brown | Chemistry |
| David Williams | NULL |
| Emma Davis | Physics |

**3. List all courses and the names of students enrolled in those courses. If a course does not have any students, show NULL for the student name.**

**Query:**

SELECT c.CourseName, s.Name AS StudentName

FROM Students s

RIGHT JOIN Courses c ON s.CourseID = c.CourseID;

**Explanation:**

* **RIGHT JOIN** returns all records from the Courses table, and matching records from the Students table.
* If there is no matching student for a course (like Computer Sci), it returns NULL for the StudentName.
* This shows all courses, even if no student is enrolled in a course (e.g., "Computer Sci").

**Result:**

| **CourseName** | **StudentName** |
| --- | --- |
| Mathematics | Alice Johnson |
| Physics | Bob Smith |
| Chemistry | Charlie Brown |
| Computer Sci | NULL |

**4. List all students and all courses. If a student is not enrolled in a course, show NULL for the course name, and if a course has no students enrolled, show NULL for the student name.**

**Query:**

SELECT s.Name AS StudentName, c.CourseName

FROM Students s

FULL JOIN Courses c ON s.CourseID = c.CourseID;

**Explanation:**

* **FULL JOIN** combines the results of both a LEFT JOIN and a RIGHT JOIN. It returns all records from both Students and Courses.
* Where there is no match (for example, a student with no course or a course with no students), it returns NULL for the unmatched side.
* This shows every student and every course, including those with no corresponding matches.

**Result:**

| **StudentName** | **CourseName** |
| --- | --- |
| Alice Johnson | Mathematics |
| Bob Smith | Physics |
| Charlie Brown | Chemistry |
| David Williams | NULL |
| Emma Davis | Physics |
| NULL | Computer Sci |

**5. List the student names and course names, assuming both the Students and Courses tables have the same CourseID column.**

**Query:**

SELECT s.Name AS StudentName, c.CourseName

FROM Students s

NATURAL JOIN Courses c;

**Explanation:**

* **NATURAL JOIN** automatically joins tables based on columns that have the same name in both tables. Here, CourseID is the common column, so it’s used for the join without explicitly specifying it.
* This type of join assumes that both tables share the same column (CourseID), and it performs the join on this column automatically.

**Result:**

| **StudentName** | **CourseName** |
| --- | --- |
| Alice Johnson | Mathematics |
| Bob Smith | Physics |
| Charlie Brown | Chemistry |
| Emma Davis | Physics |

**In RA**

**1. Find the names of students and their course names for those who are enrolled in a course.**

In **SQL**, this is essentially an **INNER JOIN**. In **relational algebra**, we use the **join (⨝)** operation.

**Relational Algebra:**

π\_Name, CourseName (Students ⨝ Courses)

**Explanation:**

* **⨝** represents the **join** operation between the Students and Courses relations, based on the matching CourseID attribute.
* **π\_Name, CourseName** selects only the Name and CourseName attributes from the resulting relation.

**2. List all students and the names of their courses. If a student is not enrolled in any course, show NULL for the course name.**

In **SQL**, this is a **LEFT JOIN**. In **relational algebra**, we use the **left outer join (⟕)**.

**Relational Algebra:**

π\_Name, CourseName (Students ⟕ Courses)

**Explanation:**

* **⟕** represents the **left outer join** between the Students and Courses relations. This ensures that all students are included, even those who are not enrolled in any course. For such students, CourseName will be NULL.

**3. List all courses and the names of students enrolled in those courses. If a course does not have any students, show NULL for the student name.**

In **SQL**, this is a **RIGHT JOIN**. In **relational algebra**, we use the **right outer join (⟖)**.

**Relational Algebra:**

scss

Copy

π\_CourseName, Name (Students ⟖ Courses)

**Explanation:**

* **⟖** represents the **right outer join** between the Students and Courses relations. This ensures that all courses are included, even those with no students enrolled. For such courses, the Name of the student will be NULL.

**4. List all students and all courses. If a student is not enrolled in a course, show NULL for the course name, and if a course has no students enrolled, show NULL for the student name.**

In **SQL**, this is a **FULL OUTER JOIN**. In **relational algebra**, we use the **full outer join (⟗)**.

**Relational Algebra:**

π\_Name, CourseName (Students ⟗ Courses)

**Explanation:**

* **⟗** represents the **full outer join** between the Students and Courses relations. This ensures that all students and all courses are listed. If a student isn't enrolled in a course, the CourseName will be NULL, and if a course has no students, the Name will be NULL.

**5. List the student names and course names, assuming both the Students and Courses tables have the same CourseID column.**

In **SQL**, this is essentially a **NATURAL JOIN**. In **relational algebra**, we can use the **natural join (⨝)**.

**Relational Algebra:**

π\_Name, CourseName (Students ⨝ Courses)

**Explanation:**

* **⨝** represents a **natural join** between the Students and Courses relations. The natural join automatically matches rows based on the common CourseID attribute and returns the Name and CourseName attributes.

| **employee\_id** | **first\_name** | **last\_name** | **email** | **phone\_number** | **job\_ID** | **salary** |
| --- | --- | --- | --- | --- | --- | --- |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 100 | John | Doe | john.doe@email.com | 1234567890 | DEV | 4500 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 101 | Jane | Smith | jane.smith@email.com | 2345678901 | DEV | 5500 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 102 | Mike | Rogers | mike.rogers@email.com | 3456789012 | MANAGER | 7000 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 103 | Mary | Johnson | mary.johnson@email.com | 4567890123 | HR | 3500 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 104 | Steve | Brown | steve.brown@email.com | 5678901234 | DEV | 6000 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 105 | Emily | Davis | emily.davis@email.com | 6789012345 | MANAGER | 8000 |

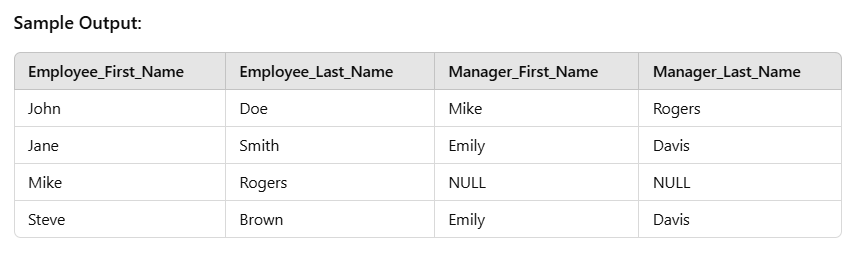
1. **Find the first and last names of employees along with their manager's first and last names.**

SELECT e.first\_name AS Employee\_First\_Name, e.last\_name AS Employee\_Last\_Name,

m.first\_name AS Manager\_First\_Name, m.last\_name AS Manager\_Last\_Name

FROM Employee e

LEFT JOIN Employee m ON e.manager\_id = m.employee\_id;

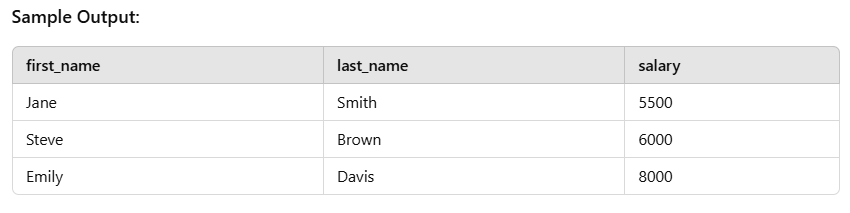


1. **Find employees who earn more than the average salary of employees in the 'DEV' job category.**

SELECT first\_name, last\_name, salary

FROM Employee

WHERE salary > (SELECT AVG(salary) FROM Employee WHERE job\_ID = 'DEV');

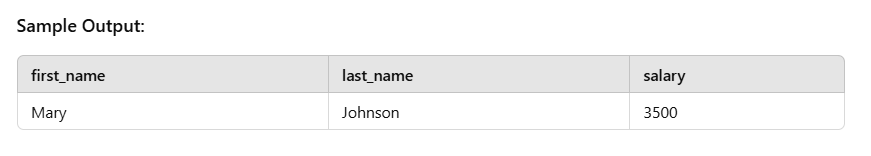
****

**3. Find employees who are in the 'HR' department and have a salary greater than the average salary in the 'MANAGER' job category.**

SELECT first\_name, last\_name, salary

FROM Employee

WHERE department\_id = 'HR' AND salary > (SELECT AVG(salary) FROM Employee WHERE job\_ID = 'MANAGER');



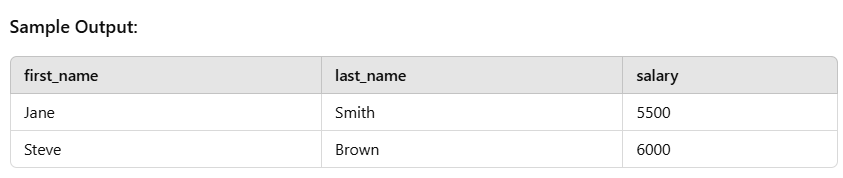
**4List the employees who report to a manager who earns more than 7000.**

SELECT e.first\_name, e.last\_name, e.salary

FROM Employee e

JOIN Employee m ON e.manager\_id = m.employee\_id

WHERE m.salary > 7000;

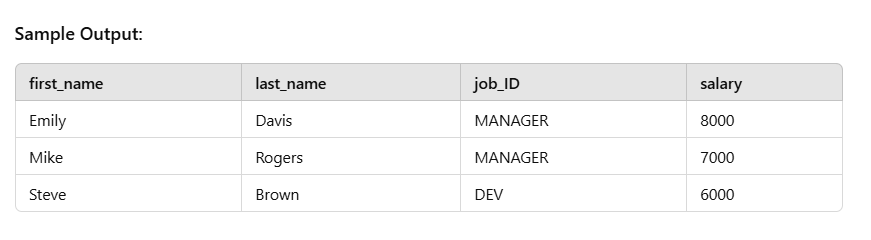


1. **Find the employee with the highest salary in each job category.**

SELECT first\_name, last\_name, job\_ID, salary

FROM Employee e

WHERE salary = (SELECT MAX(salary) FROM Employee WHERE job\_ID = e.job\_ID);

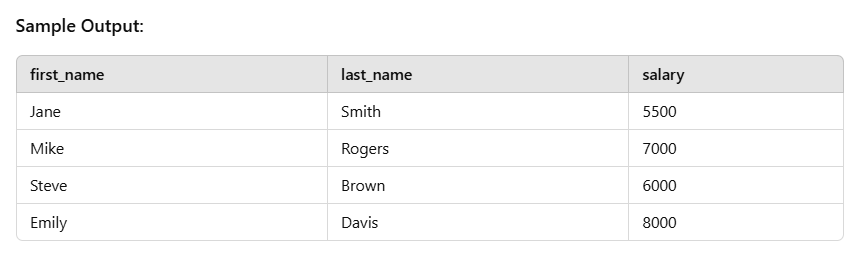


1. **Find employees who earn more than the average salary of all employees.**

SELECT first\_name, last\_name, salary

FROM Employee

WHERE salary > (SELECT AVG(salary) FROM Employee);



1. **Find the employees who have the same salary as the employee with employee\_id = 104.**

SELECT first\_name, last\_name, salary

FROM Employee

WHERE salary = (SELECT salary FROM Employee WHERE employee\_id = 104);

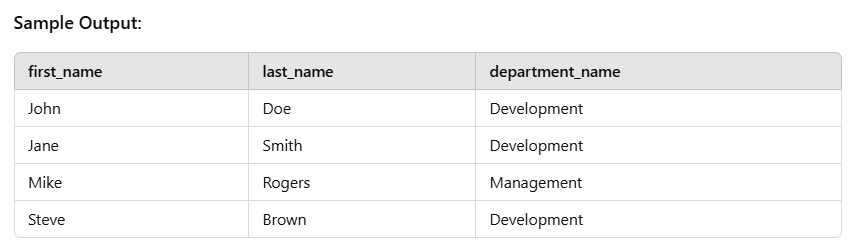


1. **List employees along with their department names.**

SELECT e.first\_name, e.last\_name, d.department\_name

FROM Employee e

JOIN Department d ON e.department\_id = d.department\_id;



1. **Find the employees whose job title is not 'MANAGER'.**

SELECT first\_name, last\_name, job\_ID

FROM Employee

WHERE job\_ID NOT IN (SELECT job\_ID FROM Employee WHERE job\_ID = 'MANAGER');

