Assume Tables and Schema

Sample Tables:

1. **Employees**: Contains employee information.
   - **Attributes**: `emp_id`, `name`, `age`, `department_id`, `salary`
2. **Departments**: Contains department details.
   - **Attributes**: `dept_id`, `dept_name`, `location`
3. **Projects**: Contains project details.
   - **Attributes**: `proj_id`, `proj_name`, `dept_id`, `budget`
4. **Works_On**: Contains details of employees working on projects.
   - **Attributes**: `emp_id`, `proj_id`, `hours`
5. **Salaries**: Contains historical salary data.
   - **Attributes**: `emp_id`, `year`, `annual_salary`

Relational Algebra Queries

Selection (σ)

```
1. σ(age > 30)(Employees)
2. σ(dept_name = 'HR')(Departments)
3. σ(salary > 70000)(Employees)
4. σ(location = 'New York')(Departments)
5. σ(budget > 100000)(Projects)
```

Projection (*π*)

```
6. π(name, age)(Employees)
7. π(dept_name)(Departments)
8. π(proj_name, budget)(Projects)
9. π(emp_id, salary)(Employees)
10.π(location)(Departments)
```

Union (∪)

```
11.π(emp_id, salary)(Employees) U π(emp_id, annual_salary)
   (Salaries)
12.π(dept_id, location)(Departments) U π(dept_id, dept_name)
   (Departments)
13.π(emp_id, proj_id)(Works_On) U π(emp_id, proj_id)(Works_On)
14.π(dept_id)(Departments) U π(dept_id)(Projects)
15.π(name)(Employees) U π(proj_name)(Projects)
```

Set Difference (−)

16. π(emp_id)(Employees) - π(emp_id)(Works_On)
17. π(dept_id)(Departments) - π(dept_id)(Projects)
18. π(proj_id)(Projects) - π(proj_id)(Works_On)
19. π(emp_id)(Employees) - π(emp_id)(Salaries)
20. π(location)(Departments) - π(location)(Departments)

Cartesian Product (×)

21. Employees × Departments
22. Projects × Departments
23. Employees × Projects
24. Departments × Works_On
25. Salaries × Works_On

Rename (ρ)

26. ρ(Emp(emp_id, name, age, dept_id, salary))(Employees)
27. ρ(Dept(dept_id, dept_name, location))(Departments)
28. ρ(Proj(proj_id, proj_name, dept_id, budget))(Projects)
29. ρ(Work(emp_id, proj_id, hours))(Works_On)
30. ρ(Sal(emp_id, year, annual_salary))(Salaries)

Join (⋈)

31. Employees ⋈ dept_id = dept_id Departments
32. Projects ⋈ dept_id = dept_id Departments
33. Works_On ⋈ emp_id = emp_id Employees
34. Employees ⋈ emp_id = emp_id Salaries
35. Projects ⋈ dept_id = dept_id Works_On

Natural Join (⋈)

36. Employees ⋈ Departments
37. Projects ⋈ Departments
38. Works_On ⋈ Employees
39. Employees ⋈ Salaries
40. Projects ⋈ Works_On

Division (÷)

41. π(emp_id, proj_id)(Works_On) ÷ π(proj_id)(Projects)
42. π(emp_id, dept_id)(Employees) ÷ π(dept_id)(Departments)

43. π(emp_id, proj_id)(Works_On) ÷ π(proj_id)(Works_On)
44. π(emp_id, proj_id)(Works_On) ÷ π(emp_id)(Employees)
45. π(proj_id)(Projects) ÷ π(dept_id)(Departments)

Intersection (∩)

46. π(emp_id)(Employees) ∩ π(emp_id)(Works_On)
47. π(dept_id)(Departments) ∩ π(dept_id)(Projects)
48. π(proj_id)(Projects) ∩ π(proj_id)(Works_On)
49. π(emp_id)(Employees) ∩ π(emp_id)(Salaries)
50. π(location)(Departments) ∩ π(location)(Departments)

SQL Queries

Basic SELECT

1. `SELECT * FROM Employees WHERE age > 30;`
2. `SELECT * FROM Departments WHERE dept_name = 'HR';`
3. `SELECT * FROM Employees WHERE salary > 70000;`
4. `SELECT * FROM Departments WHERE location = 'New York';`
5. `SELECT * FROM Projects WHERE budget > 100000;`

SELECT with Specific Columns

6. `SELECT name, age FROM Employees;`
7. `SELECT dept_name FROM Departments;`
8. `SELECT proj_name, budget FROM Projects;`
9. `SELECT emp_id, salary FROM Employees;`
10. `SELECT location FROM Departments;`

UNION

11. `SELECT emp_id, salary FROM Employees UNION SELECT emp_id, annual_salary FROM Salaries;`
12. `SELECT dept_id, location FROM Departments UNION SELECT dept_id, dept_name FROM Departments;`
13. `SELECT emp_id, proj_id FROM Works_On UNION SELECT emp_id, proj_id FROM Works_On;`
14. `SELECT dept_id FROM Departments UNION SELECT dept_id FROM Projects;`
15. `SELECT name FROM Employees UNION SELECT proj_name FROM Projects;`

EXCEPT (Set Difference)

16. `SELECT emp_id FROM Employees EXCEPT SELECT emp_id FROM Works_On;`
17. `SELECT dept_id FROM Departments EXCEPT SELECT dept_id FROM Projects;`
18. `SELECT proj_id FROM Projects EXCEPT SELECT proj_id FROM Works_On;`
19. `SELECT emp_id FROM Employees EXCEPT SELECT emp_id FROM Salaries;`
20. `SELECT location FROM Departments EXCEPT SELECT location FROM Departments;`

JOIN

21. `SELECT * FROM Employees INNER JOIN Departments ON Employees.department_id = Departments.dept_id;`
22. `SELECT * FROM Projects INNER JOIN Departments ON Projects.dept_id = Departments.dept_id;`
23. `SELECT * FROM Works_On INNER JOIN Employees ON Works_On.emp_id = Employees.emp_id;`
24. `SELECT * FROM Employees INNER JOIN Salaries ON Employees.emp_id = Salaries.emp_id;`
25. `SELECT * FROM Projects INNER JOIN Works_On ON Projects.proj_id = Works_On.proj_id;`

LEFT JOIN

26. `SELECT * FROM Employees LEFT JOIN Departments ON Employees.department_id = Departments.dept_id;`
27. `SELECT * FROM Projects LEFT JOIN Departments ON Projects.dept_id = Departments.dept_id;`
28. `SELECT * FROM Works_On LEFT JOIN Employees ON Works_On.emp_id = Employees.emp_id;`
29. `SELECT * FROM Employees LEFT JOIN Salaries ON Employees.emp_id = Salaries.emp_id;`
30. `SELECT * FROM Projects LEFT JOIN Works_On ON Projects.proj_id = Works_On.proj_id;`

RIGHT JOIN

31. `SELECT * FROM Employees RIGHT JOIN Departments ON Employees.department_id = Departments.dept_id;`
32. `SELECT * FROM Projects RIGHT JOIN Departments ON Projects.dept_id = Departments.dept_id;`
33. `SELECT * FROM Works_On RIGHT JOIN Employees ON Works_On.emp_id = Employees.emp_id;`
34. `SELECT * FROM Employees RIGHT JOIN Salaries ON Employees.emp_id = Salaries.emp_id;`
35. `SELECT * FROM Projects RIGHT JOIN Works_On ON Projects.proj_id = Works_On.proj_id;`

FULL OUTER JOIN

36. SELECT * FROM Employees FULL OUTER JOIN Departments ON Employees.department_id = Departments.dept_id;
37. SELECT * FROM Projects FULL OUTER JOIN Departments ON Projects.dept_id = Departments.dept_id;
38. SELECT * FROM Works_On FULL OUTER JOIN Employees ON Works_On.emp_id = Employees.emp_id;
39. SELECT * FROM Employees FULL OUTER JOIN Salaries ON Employees.emp_id = Salaries.emp_id;
40. SELECT * FROM Projects FULL OUTER JOIN Works_On ON Projects.proj_id = Works_On.proj_id;

Subqueries

41. SELECT name FROM Employees WHERE emp_id IN (SELECT emp_id FROM Works_On WHERE proj_id = 101);
42. SELECT dept_name FROM Departments WHERE dept_id = (SELECT dept_id FROM Projects WHERE proj_name = 'Project A');
43. SELECT emp_id FROM Employees WHERE salary = (SELECT MAX(salary) FROM Employees);
44. SELECT proj_id FROM Projects WHERE dept_id = (SELECT dept_id FROM Departments WHERE dept_name = 'Finance');
45. SELECT emp_id FROM Works_On WHERE proj_id IN (SELECT proj_id FROM Projects WHERE budget > 50000);

Aggregation

46. SELECT AVG(salary) FROM Employees;
47. SELECT COUNT(emp_id) FROM Employees;
48. SELECT SUM(budget) FROM Projects;
49. SELECT MIN(age) FROM Employees;

Descriptions of Relational Algebra (RA) Queries

Selection (σ)

1. **σ(age > 30)(Employees)**: Selects all employees older than 30 years.
   - **Minimum Tuples**: 0 (if no employee is older than 30)
   - **Maximum Tuples**: Total number of tuples in `Employees`
2. **σ(dept_name = 'HR')(Departments)**: Selects all departments named 'HR'.
   - **Minimum Tuples**: 0 (if there is no 'HR' department)
   - **Maximum Tuples**: Total number of tuples in `Departments`
3. **σ(salary > 70000)(Employees)**: Selects all employees with a salary greater than 70,000.
   - **Minimum Tuples**: 0 (if no employee earns more than 70,000)
   - **Maximum Tuples**: Total number of tuples in `Employees`
4. **σ(location = 'New York')(Departments)**: Selects all departments located in New York.
   - **Minimum Tuples**: 0 (if no department is in New York)
   - **Maximum Tuples**: Total number of tuples in `Departments`
5. **σ(budget > 100000)(Projects)**: Selects all projects with a budget greater than 100,000.
   - **Minimum Tuples**: 0 (if no project has a budget above 100,000)
   - **Maximum Tuples**: Total number of tuples in `Projects`

Projection (*π*)

6. **π(name, age)(Employees)**: Projects the names and ages of all employees.
   - **Minimum Tuples**: 0 (if `Employees` is empty)
   - **Maximum Tuples**: Total number of tuples in `Employees`
7. **π(dept_name)(Departments)**: Projects the names of all departments.
   - **Minimum Tuples**: 0 (if `Departments` is empty)
   - **Maximum Tuples**: Total number of tuples in `Departments`
8. **π(proj_name, budget)(Projects)**: Projects the names and budgets of all projects.
   - **Minimum Tuples**: 0 (if `Projects` is empty)
   - **Maximum Tuples**: Total number of tuples in `Projects`
9. **π(emp_id, salary)(Employees)**: Projects the employee IDs and salaries.
   - **Minimum Tuples**: 0 (if `Employees` is empty)
   - **Maximum Tuples**: Total number of tuples in `Employees`
10. **π(location)(Departments)**: Projects the locations of all departments.
    - **Minimum Tuples**: 0 (if `Departments` is empty)
    - **Maximum Tuples**: Total number of tuples in `Departments`

Union (∪)

11. **π(emp_id, salary)(Employees) U π(emp_id, annual_salary)(Salaries)**: Combines employee IDs and their current or annual salaries.
    - **Minimum Tuples**: Size of the larger relation (whichever has more unique `emp_id`)
    - **Maximum Tuples**: Sum of unique tuples from both `Employees` and `Salaries`
12. **π(dept_id, location)(Departments) U π(dept_id, dept_name)(Departments)**: Combines department IDs and their respective locations or names.
    - **Minimum Tuples**: Size of the `Departments` table (assuming all entries are unique)
    - **Maximum Tuples**: Sum of unique tuples from both operations on `Departments`
13. **π(emp_id, proj_id)(Works_On) U π(emp_id, proj_id)(Works_On)**: Combines employees and project IDs they work on. This will likely be redundant.
    - **Minimum Tuples**: Size of `Works_On` (assuming duplicates are removed)
    - **Maximum Tuples**: Size of `Works_On` (if there are no duplicates)
14. **π(dept_id)(Departments) U π(dept_id)(Projects)**: Combines department IDs from departments and projects.
    - **Minimum Tuples**: Number of unique `dept_id` in `Departments` or `Projects`, whichever is larger
    - **Maximum Tuples**: Sum of unique `dept_id` in `Departments` and `Projects`
15. **π(name)(Employees) U π(proj_name)(Projects)**: Combines employee names and project names into a single set.
    - **Minimum Tuples**: Number of unique names in `Employees` or `Projects`, whichever is larger
    - **Maximum Tuples**: Sum of unique names in `Employees` and `Projects`

Set Difference (−)

16. **π(emp_id)(Employees) - π(emp_id)(Works_On)**: Finds employees not working on any project.
    - **Minimum Tuples**: 0 (if all employees work on at least one project)
    - **Maximum Tuples**: Total number of tuples in `Employees`
17. **π(dept_id)(Departments) - π(dept_id)(Projects)**: Finds departments that have no associated projects.
    - **Minimum Tuples**: 0 (if all departments have projects)
    - **Maximum Tuples**: Total number of tuples in `Departments`
18. **π(proj_id)(Projects) - π(proj_id)(Works_On)**: Finds projects that have no employees working on them.
    - **Minimum Tuples**: 0 (if all projects have employees working on them)
    - **Maximum Tuples**: Total number of tuples in `Projects`
19. **π(emp_id)(Employees) - π(emp_id)(Salaries)**: Finds employees without any salary record.
    - **Minimum Tuples**: 0 (if all employees have salary records)

    - **Maximum Tuples**: Total number of tuples in `Employees`
20. **π(location)(Departments) - π(location)(Departments)**: Finds locations that are not used in any department. This will always result in an empty set because a set minus itself is empty.
    - **Minimum Tuples**: 0
    - **Maximum Tuples**: 0

Cartesian Product (×)

21. **Employees × Departments**: Creates a Cartesian product of all employees with all departments.
    - **Minimum Tuples**: 0 (if either `Employees` or `Departments` is empty)
    - **Maximum Tuples**: Product of the number of tuples in `Employees` and `Departments`
22. **Projects × Departments**: Creates a Cartesian product of all projects with all departments.
    - **Minimum Tuples**: 0 (if either `Projects` or `Departments` is empty)
    - **Maximum Tuples**: Product of the number of tuples in `Projects` and `Departments`
23. **Employees × Projects**: Creates a Cartesian product of all employees with all projects.
    - **Minimum Tuples**: 0 (if either `Employees` or `Projects` is empty)
    - **Maximum Tuples**: Product of the number of tuples in `Employees` and `Projects`
24. **Departments × Works_On**: Creates a Cartesian product of all departments with all project assignments.
    - **Minimum Tuples**: 0 (if either `Departments` or `Works_On` is empty)
    - **Maximum Tuples**: Product of the number of tuples in `Departments` and `Works_On`
25. **Salaries × Works_On**: Creates a Cartesian product of all salaries with all project assignments.
    - **Minimum Tuples**: 0 (if either `Salaries` or `Works_On` is empty)
    - **Maximum Tuples**: Product of the number of tuples in `Salaries` and `Works_On`

Rename (ρ)

26. **ϱ(Emp(emp_id, name, age, dept_id, salary))(Employees)**: Renames the `Employees` relation to `Emp`.
    - **Minimum Tuples**: 0 (if `Employees` is empty)
    - **Maximum Tuples**: Total number of tuples in `Employees`
27. **ϱ(Dept(dept_id, dept_name, location))(Departments)**: Renames the `Departments` relation to `Dept`.
    - **Minimum Tuples**: 0 (if `Departments` is empty)
    - **Maximum Tuples**: Total number of tuples in `Departments`
28. **ϱ(Proj(proj_id, proj_name, dept_id, budget))(Projects)**: Renames the `Projects` relation to `Proj`.

   - **Minimum Tuples**: 0 (if `Projects` is empty)
   - **Maximum Tuples**: Total number of tuples in `Projects`
29. **ϱ(Work(emp_id, proj_id, hours))(Works_On)**: Renames the `Works_On` relation to `Work`.
   - **Minimum Tuples**: 0 (if `Works_On` is empty)
   - **Maximum Tuples**: Total number of tuples in `Works_On`
30. **ϱ(Sal(emp_id, year, annual_salary))(Salaries)**: Renames the `Salaries` relation to `Sal`.
   - **Minimum Tuples**: 0 (if `Salaries` is empty)
   - **Maximum Tuples**: Total number of tuples in `Salaries`

Join (⋈) (continued)

31. **Employees ⋈ dept_id = dept_id Departments**: Joins employees with their respective departments on `dept_id`.
- **Minimum Tuples**: 0 (if no `dept_id` matches between `Employees` and `Departments`)
- **Maximum Tuples**: Total number of tuples in `Employees` (if every employee belongs to a department)
32. **Projects ⋈ dept_id = dept_id Departments**: Joins projects with their respective departments on `dept_id`.
- **Minimum Tuples**: 0 (if no `dept_id` matches between `Projects` and `Departments`)
- **Maximum Tuples**: Total number of tuples in `Projects` (if every project is associated with a department)
33. **Works_On ⋈ emp_id = emp_id Employees**: Joins `Works_On` with `Employees` on `emp_id` to get detailed employee information.
- **Minimum Tuples**: 0 (if no `emp_id` matches between `Works_On` and `Employees`)
- **Maximum Tuples**: Total number of tuples in `Works_On` (if every entry corresponds to a valid employee)
34. **Employees ⋈ emp_id = emp_id Salaries**: Joins `Employees` with `Salaries` on `emp_id` to get historical salary data.
- **Minimum Tuples**: 0 (if no `emp_id` matches between `Employees` and `Salaries`)
- **Maximum Tuples**: Total number of tuples in `Salaries` (if every salary record corresponds to a valid employee)
35. **Projects ⋈ dept_id = dept_id Works_On**: This join does not make logical sense as `Projects` should ideally join with `Works_On` on `proj_id`, not `dept_id`. Assuming it was meant to be `proj_id`:
- **Minimum Tuples**: 0 (if no `proj_id` matches between `Projects` and `Works_On`)
- **Maximum Tuples**: Total number of tuples in `Works_On` (if every work assignment corresponds to a valid project)

Natural Join (⋈)

36. **Employees ⋈ Departments**: Performs a natural join between `Employees` and `Departments` on `dept_id`.
- **Minimum Tuples**: 0 (if no common `dept_id` exists)
- **Maximum Tuples**: Number of tuples in `Employees` (if all employees belong to valid departments)
37. **Projects ⋈ Departments**: Performs a natural join between `Projects` and `Departments` on `dept_id`.

- **Minimum Tuples**: 0 (if no common `dept_id` exists)
- **Maximum Tuples**: Number of tuples in `Projects` (if all projects are linked to valid departments)

38. **Works_On ⋈ Employees**: Performs a natural join on `emp_id` between `Works_On` and `Employees`.
- **Minimum Tuples**: 0 (if no common `emp_id` exists)
- **Maximum Tuples**: Number of tuples in `Works_On` (if all work assignments correspond to valid employees)

39. **Employees ⋈ Salaries**: Performs a natural join on `emp_id` between `Employees` and `Salaries`.
- **Minimum Tuples**: 0 (if no common `emp_id` exists)
- **Maximum Tuples**: Number of tuples in `Salaries` (if all salary records correspond to valid employees)

40. **Projects ⋈ Works_On**: Performs a natural join on `proj_id` between `Projects` and `Works_On`.
- **Minimum Tuples**: 0 (if no common `proj_id` exists)
- **Maximum Tuples**: Number of tuples in `Works_On` (if all work assignments are linked to valid projects)

Division (÷)

41. **π(emp_id, proj_id)(Works_On) ÷ π(proj_id)(Projects)**: Finds employees who have worked on all projects listed.
- **Minimum Tuples**: 0 (if no employee has worked on all projects)
- **Maximum Tuples**: Number of unique `emp_id` in `Works_On` (if every employee has worked on all projects)

42. **π(emp_id, dept_id)(Employees) ÷ π(dept_id)(Departments)**: Finds employees assigned to all departments.
- **Minimum Tuples**: 0 (if no employee is in all departments)
- **Maximum Tuples**: Number of unique `emp_id` in `Employees` (if every employee is in all departments)

43. **π(emp_id, proj_id)(Works_On) ÷ π(proj_id)(Works_On)**: This seems to be a division operation of `Works_On` by itself, which would result in an empty relation unless there is an implied constant or universal project set.
- **Minimum Tuples**: 0
- **Maximum Tuples**: Typically 0, as the division of a set by itself should yield the universal set unless constrained

44. **π(emp_id, proj_id)(Works_On) ÷ π(emp_id)(Employees)**: This operation checks if all employees are working on a particular set of projects. Typically, division requires consistent sets on both sides.
- **Minimum Tuples**: 0 (if the sets do not align)
- **Maximum Tuples**: Number of unique `emp_id` in `Works_On`

45. **π(proj_id)(Projects) ÷ π(dept_id)(Departments)**: Division here seems incorrect as projects and departments should not divide over each other logically. Assuming it checks projects per department:
- **Minimum Tuples**: 0 (if no project exists for all departments)
- **Maximum Tuples**: Number of unique `proj_id` (but typically not valid)

Intersection (∩)

46. **π(emp_id)(Employees) ∩ π(emp_id)(Works_On)**: Finds employees who are also listed in the `Works_On` table.
- **Minimum Tuples**: 0 (if no common `emp_id` exists)
- **Maximum Tuples**: Number of unique `emp_id` in `Employees` or `Works_On`, whichever is smaller

47. **π(dept_id)(Departments) ∩ π(dept_id)(Projects)**: Finds department IDs common to both departments and projects.
- **Minimum Tuples**: 0 (if no common `dept_id` exists)
- **Maximum Tuples**: Number of unique `dept_id` in `Departments` or `Projects`, whichever is smaller

48. **π(proj_id)(Projects) ∩ π(proj_id)(Works_On)**: Finds project IDs that exist in both the `Projects` and `Works_On` tables.
- **Minimum Tuples**: 0 (if no common `proj_id` exists)
- **Maximum Tuples**: Number of unique `proj_id` in `Projects` or `Works_On`, whichever is smaller

49. **π(emp_id)(Employees) ∩ π(emp_id)(Salaries)**: Finds employees with records in both `Employees` and `Salaries`.
- **Minimum Tuples**: 0 (if no common `emp_id` exists)
- **Maximum Tuples**: Number of unique `emp_id` in `Employees` or `Salaries`, whichever is smaller

50. **π(location)(Departments) ∩ π(location)(Departments)**: This operation is redundant and will simply return all unique department locations.
- **Minimum Tuples**: Number of unique `location` in `Departments`
- **Maximum Tuples**: Number of unique `location` in `Departments`

Corresponding SQL:

Corresponding SQL Queries :

Basic SELECT

1. **`SELECT * FROM Employees WHERE age > 30;`**: Retrieves all columns for employees older than 30.
   - **Minimum Tuples**: 0 (if no employee is older than 30)
   - **Maximum Tuples**: Total number of rows in `Employees`
2. **`SELECT * FROM Departments WHERE dept_name = 'HR';`**: Retrieves all columns for departments named 'HR'.
   - **Minimum Tuples**: 0 (if there is no 'HR' department)
   - **Maximum Tuples**: Total number of rows in `Departments`
3. **`SELECT * FROM Employees WHERE salary > 70000;`**: Retrieves all employees with a salary over 70,000.
   - **Minimum Tuples**: 0 (if no employee earns more than 70,000)
   - **Maximum Tuples**: Total number of rows in `Employees`
4. **`SELECT * FROM Departments WHERE location = 'New York';`**: Retrieves all departments located in New York.

  - **Minimum Tuples**: 0 (if no department is in New York)
  - **Maximum Tuples**: Total number of rows in `Departments`
5. **`SELECT * FROM Projects WHERE budget > 100000;`**: Retrieves all projects with a budget over 100,000.
  - **Minimum Tuples**: 0 (if no project has a budget above 100,000)
  - **Maximum Tuples**: Total number of rows in `Projects`

SELECT with Specific Columns (continued)

6. **`SELECT name, age FROM Employees;`**: Retrieves only the `name` and `age` columns for all employees.
  - **Minimum Tuples**: 0 (if `Employees` table is empty)
  - **Maximum Tuples**: Total number of rows in `Employees`
7. **`SELECT dept_name FROM Departments;`**: Retrieves only the `dept_name` column for all departments.
  - **Minimum Tuples**: 0 (if `Departments` table is empty)
  - **Maximum Tuples**: Total number of rows in `Departments`
8. **`SELECT proj_name, budget FROM Projects;`**: Retrieves the `proj_name` and `budget` columns for all projects.
  - **Minimum Tuples**: 0 (if `Projects` table is empty)
  - **Maximum Tuples**: Total number of rows in `Projects`
9. **`SELECT emp_id, salary FROM Employees;`**: Retrieves the `emp_id` and `salary` columns for all employees.
  - **Minimum Tuples**: 0 (if `Employees` table is empty)
  - **Maximum Tuples**: Total number of rows in `Employees`
10. **`SELECT location FROM Departments;`**: Retrieves the `location` column for all departments.
  - **Minimum Tuples**: 0 (if `Departments` table is empty)
  - **Maximum Tuples**: Total number of rows in `Departments`

JOIN Operations

11. **`SELECT Employees.name, Departments.dept_name FROM Employees JOIN Departments ON Employees.dept_id = Departments.dept_id;`**: Joins `Employees` and `Departments` on `dept_id` and selects employee names and department names.
  - **Minimum Tuples**: 0 (if no matching `dept_id` exists between `Employees` and `Departments`)
  - **Maximum Tuples**: Total number of rows in `Employees` (if every employee is associated with a valid department)
12. **`SELECT Projects.proj_name, Departments.location FROM Projects JOIN Departments ON Projects.dept_id = Departments.dept_id;`**: Joins `Projects` and `Departments` on `dept_id` and selects project names and department locations.

   - **Minimum Tuples**: 0 (if no matching `dept_id` exists between `Projects` and `Departments`)
   - **Maximum Tuples**: Total number of rows in `Projects` (if every project is associated with a valid department)
13. **`SELECT Employees.emp_id, Works_On.proj_id FROM Employees JOIN Works_On ON Employees.emp_id = Works_On.emp_id;`**: Joins `Employees` and `Works_On` on `emp_id` and selects employee IDs and project IDs.

   - **Minimum Tuples**: 0 (if no matching `emp_id` exists between `Employees` and `Works_On`)
   - **Maximum Tuples**: Total number of rows in `Works_On` (if every project assignment has a corresponding employee)
14. **`SELECT Employees.emp_id, Salaries.annual_salary FROM Employees JOIN Salaries ON Employees.emp_id = Salaries.emp_id;`**: Joins `Employees` and `Salaries` on `emp_id` and selects employee IDs and annual salaries.

   - **Minimum Tuples**: 0 (if no matching `emp_id` exists between `Employees` and `Salaries`)
   - **Maximum Tuples**: Total number of rows in `Salaries` (if every salary record corresponds to a valid employee)
15. **`SELECT Projects.proj_id, Works_On.emp_id FROM Projects JOIN Works_On ON Projects.proj_id = Works_On.proj_id;`**: Joins `Projects` and `Works_On` on `proj_id` and selects project IDs and employee IDs.

   - **Minimum Tuples**: 0 (if no matching `proj_id` exists between `Projects` and `Works_On`)
   - **Maximum Tuples**: Total number of rows in `Works_On` (if every work assignment corresponds to a valid project)

UNION Operations

16. **`SELECT emp_id, salary FROM Employees UNION SELECT emp_id, annual_salary FROM Salaries;`**: Combines employee IDs and their current or annual salaries.

   - **Minimum Tuples**: Number of unique `emp_id` across `Employees` and `Salaries`
   - **Maximum Tuples**: Sum of unique `emp_id` tuples in both `Employees` and `Salaries`
17. **`SELECT dept_id, location FROM Departments UNION SELECT dept_id, dept_name FROM Departments;`**: Combines department IDs with their respective locations or names.

   - **Minimum Tuples**: Number of unique `dept_id` in `Departments` (assuming unique pairs)
   - **Maximum Tuples**: Sum of unique tuples from both queries on `Departments`
18. **`SELECT emp_id, proj_id FROM Works_On UNION SELECT emp_id, proj_id FROM Works_On;`**: Combines employee and project IDs they work on, but since it's the same relation, it results in unique tuples.

- **Minimum Tuples**: Number of unique `(emp_id, proj_id)` pairs in `Works_On`
- **Maximum Tuples**: Number of unique `(emp_id, proj_id)` pairs in `Works_On`

19. **`SELECT dept_id FROM Departments UNION SELECT dept_id FROM Projects;`**: Combines department IDs from departments and projects.

    - **Minimum Tuples**: Number of unique `dept_id` in `Departments` or `Projects`, whichever is larger
    - **Maximum Tuples**: Sum of unique `dept_id` tuples in both `Departments` and `Projects`

20. **`SELECT name FROM Employees UNION SELECT proj_name FROM Projects;`**: Combines employee names and project names.

    - **Minimum Tuples**: Number of unique names in `Employees` or `Projects`, whichever is larger
    - **Maximum Tuples**: Sum of unique names in both `Employees` and `Projects`

INTERSECT Operations

21. **`SELECT emp_id FROM Employees INTERSECT SELECT emp_id FROM Works_On;`**: Finds employee IDs that are present in both `Employees` and `Works_On`.

    - **Minimum Tuples**: 0 (if no `emp_id` is common)
    - **Maximum Tuples**: Number of unique `emp_id` in `Employees` or `Works_On`, whichever is smaller

22. **`SELECT dept_id FROM Departments INTERSECT SELECT dept_id FROM Projects;`**: Finds department IDs that are present in both `Departments` and `Projects`.

    - **Minimum Tuples**: 0 (if no `dept_id` is common)
    - **Maximum Tuples**: Number of unique `dept_id` in `Departments` or `Projects`, whichever is smaller

23. **`SELECT proj_id FROM Projects INTERSECT SELECT proj_id FROM Works_On;`**: Finds project IDs that are present in both `Projects` and `Works_On`.

    - **Minimum Tuples**: 0 (if no `proj_id` is common)
    - **Maximum Tuples**: Number of unique `proj_id` in `Projects` or `Works_On`, whichever is smaller

24. **`SELECT emp_id FROM Employees INTERSECT SELECT emp_id FROM Salaries;`**: Finds employee IDs that are present in both `Employees` and `Salaries`.

    - **Minimum Tuples**: 0 (if no `emp_id` is common)
    - **Maximum Tuples**: Number of unique `emp_id` in `Employees` or `Salaries`, whichever is smaller

25. **`SELECT location FROM Departments INTERSECT SELECT location FROM Departments;`**: This operation is redundant and returns all unique department locations.

    - **Minimum Tuples**: Number of unique `location` in `Departments`
    - **Maximum Tuples**: Number of unique `location` in `Departments`

MINUS (EXCEPT) Operations

26. **`SELECT emp_id FROM Employees EXCEPT SELECT emp_id FROM Works_On;`**: Finds employee IDs that are in `Employees` but not in `Works_On`.
    - **Minimum Tuples**: 0 (if all employees have a project record in `Works_On`)
    - **Maximum Tuples**: Number of unique `emp_id` in `Employees`
27. **`SELECT dept_id FROM Departments EXCEPT SELECT dept_id FROM Projects;`**: Finds department IDs that are in `Departments` but not associated with any project in `Projects`.
    - **Minimum Tuples**: 0 (if every department is associated with a project)
    - **Maximum Tuples**: Number of unique `dept_id` in `Departments`
28. **`SELECT proj_id FROM Projects EXCEPT SELECT proj_id FROM Works_On;`**: Finds project IDs that are in `Projects` but have no employees working on them in `Works_On`.
    - **Minimum Tuples**: 0 (if every project has employees working on it)
    - **Maximum Tuples**: Number of unique `proj_id` in `Projects`

`29. SELECT emp_id FROM Employees EXCEPT SELECT emp_id FROM Salaries;`: Finds employee IDs that are in `Employees` but have no salary record in `Salaries`.

- **Minimum Tuples**: 0 (if every employee has a salary record)
- **Maximum Tuples**: Number of unique `emp_id` in `Employees`

GROUP BY and HAVING (continued)

31. **`SELECT dept_id, COUNT(emp_id) FROM Employees GROUP BY dept_id;`**: Counts the number of employees in each department.
- **Minimum Tuples**: 0 (if `Employees` table is empty)
- **Maximum Tuples**: Number of unique `dept_id` in `Employees`
32. **`SELECT proj_id, SUM(budget) FROM Projects GROUP BY proj_id;`**: Sums the budgets of projects, grouped by project ID.
- **Minimum Tuples**: 0 (if `Projects` table is empty)
- **Maximum Tuples**: Number of unique `proj_id` in `Projects`
33. **`SELECT location, COUNT(dept_id) FROM Departments GROUP BY location;`**: Counts the number of departments in each location.
- **Minimum Tuples**: 0 (if `Departments` table is empty)
- **Maximum Tuples**: Number of unique `location` in `Departments`
34. **`SELECT dept_id, AVG(salary) FROM Employees GROUP BY dept_id HAVING AVG(salary) > 70000;`**: Calculates the average salary per department and only returns departments with an average salary greater than 70,000.
- **Minimum Tuples**: 0 (if no department's average salary exceeds 70,000)
- **Maximum Tuples**: Number of unique `dept_id` in `Employees`
35. **`SELECT proj_id, COUNT(emp_id) FROM Works_On GROUP BY proj_id HAVING COUNT(emp_id) > 5;`**: Counts the number of employees working on each project and returns only projects with more than 5 employees.

- **Minimum Tuples**: 0 (if no project has more than 5 employees)
- **Maximum Tuples**: Number of unique `proj_id` in `Works_On`

Aggregation Functions

36. **`SELECT COUNT(*) FROM Employees;`**: Counts the total number of employees.
- **Minimum Tuples**: 0 (if `Employees` table is empty)
- **Maximum Tuples**: 1 (always returns a single count)

37. **`SELECT AVG(salary) FROM Employees;`**: Calculates the average salary of all employees.
- **Minimum Tuples**: 0 (if `Employees` table is empty)
- **Maximum Tuples**: 1 (always returns a single average value)

38. **`SELECT MAX(salary) FROM Employees;`**: Finds the maximum salary among all employees.
- **Minimum Tuples**: 0 (if `Employees` table is empty)
- **Maximum Tuples**: 1 (always returns a single maximum value)

39. **`SELECT MIN(salary) FROM Employees;`**: Finds the minimum salary among all employees.
- **Minimum Tuples**: 0 (if `Employees` table is empty)
- **Maximum Tuples**: 1 (always returns a single minimum value)

40. **`SELECT SUM(budget) FROM Projects;`**: Sums the budgets of all projects.
- **Minimum Tuples**: 0 (if `Projects` table is empty)
- **Maximum Tuples**: 1 (always returns a single sum value)

Subqueries

41. **`SELECT * FROM Employees WHERE salary > (SELECT AVG(salary) FROM Employees);`**: Retrieves all employees with a salary above the average salary.
- **Minimum Tuples**: 0 (if no employee's salary is above the average)
- **Maximum Tuples**: Total number of rows in `Employees` (if all employees earn above the average)

42. **`SELECT * FROM Projects WHERE budget > (SELECT MAX(budget) FROM Projects) - 50000;`**: Retrieves all projects with a budget greater than the maximum budget minus 50,000.
- **Minimum Tuples**: 0 (if no project meets the criteria)
- **Maximum Tuples**: Total number of rows in `Projects`

43. **`SELECT * FROM Employees WHERE dept_id = (SELECT dept_id FROM Departments WHERE dept_name = 'HR');`**: Retrieves all employees in the HR department.
- **Minimum Tuples**: 0 (if there are no employees in the HR department)
- **Maximum Tuples**: Total number of rows in `Employees`

44. **`SELECT * FROM Employees WHERE emp_id IN (SELECT emp_id FROM Works_On WHERE proj_id = 'P123');`**: Retrieves all employees who are working on project `P123`.
- **Minimum Tuples**: 0 (if no employee works on `P123`)
- **Maximum Tuples**: Total number of rows in `Employees`

45. **`SELECT * FROM Projects WHERE dept_id IN (SELECT dept_id FROM Departments WHERE location = 'New York');`**: Retrieves all projects associated with departments located in New York.

- **Minimum Tuples**: 0 (if no projects are associated with New York departments)
- **Maximum Tuples**: Total number of rows in `Projects`

EXISTS and NOT EXISTS

46. **`SELECT * FROM Employees WHERE EXISTS (SELECT 1 FROM Works_On WHERE Employees.emp_id = Works_On.emp_id);`**: Retrieves all employees who are assigned to at least one project.
- **Minimum Tuples**: 0 (if no employee is assigned to any project)
- **Maximum Tuples**: Total number of rows in `Employees`
47. **`SELECT * FROM Departments WHERE NOT EXISTS (SELECT 1 FROM Projects WHERE Departments.dept_id = Projects.dept_id);`**: Retrieves all departments that do not have any associated projects.
- **Minimum Tuples**: 0 (if every department has at least one project)
- **Maximum Tuples**: Total number of rows in `Departments`
48. **`SELECT * FROM Projects WHERE EXISTS (SELECT 1 FROM Works_On WHERE Projects.proj_id = Works_On.proj_id);`**: Retrieves all projects that have employees working on them.
- **Minimum Tuples**: 0 (if no project has employees working on it)
- **Maximum Tuples**: Total number of rows in `Projects`
49. **`SELECT * FROM Employees WHERE NOT EXISTS (SELECT 1 FROM Salaries WHERE Employees.emp_id = Salaries.emp_id);`**: Retrieves all employees who do not have a salary record.
- **Minimum Tuples**: 0 (if every employee has a salary record)
- **Maximum Tuples**: Total number of rows in `Employees`
50. **`SELECT * FROM Departments WHERE EXISTS (SELECT 1 FROM Employees WHERE Departments.dept_id = Employees.dept_id);`**: Retrieves all departments that have employees.
- **Minimum Tuples**: 0 (if no department has employees)
- **Maximum Tuples**: Total number of rows in `Departments`

Complex Queries with Multiple Clauses

51. **`SELECT dept_name, AVG(salary) FROM Employees JOIN Departments ON Employees.dept_id = Departments.dept_id GROUP BY dept_name HAVING AVG(salary) > 70000;`**: Calculates the average salary per department and returns departments with an average salary greater than 70,000.
- **Minimum Tuples**: 0 (if no department has an average salary over 70,000)
- **Maximum Tuples**: Number of unique `dept_name` in `Departments`
52. **`SELECT emp_id, COUNT(proj_id) FROM Works_On GROUP BY emp_id HAVING COUNT(proj_id) > 3;`**: Finds employees working on more than 3 projects.
- **Minimum Tuples**: 0 (if no employee works on more than 3 projects)
- **Maximum Tuples**: Number of unique `emp_id` in `Works_On`
53. **`SELECT proj_id, MAX(budget) FROM Projects GROUP BY proj_id HAVING MAX(budget) > 100000;`**: Finds projects with a budget greater than 100,000.
- **Minimum Tuples**: 0 (if no project has a budget over 100,000)
- **Maximum Tuples**: Number of unique `proj_id` in `Projects`

54. **`SELECT dept_name, COUNT(emp_id) FROM Departments JOIN Employees ON Departments.dept_id = Employees.dept_id GROUP BY dept_name;`**: Counts the number of employees in each department.

- **Minimum Tuples**: 0 (if no department has employees)
- **Maximum Tuples**: Number of unique `dept_name` in `Departments`

55. **`SELECT proj_name, SUM(budget) FROM Projects GROUP BY proj_name HAVING SUM(budget) > 500000;`**: Sums the budget of projects and returns those with a total budget greater than 500,000.

- **Minimum Tuples**: 0 (if no project's total budget exceeds 500,000)
- **Maximum Tuples**: Number of unique `proj_name` in `Projects`

Queries with ORDER BY

56. **`SELECT * FROM Employees ORDER BY salary DESC;`**: Retrieves all employees, ordered by salary in descending order.

- **Minimum Tuples**: 0 (if `Employees` table is empty)
- **Maximum Tuples**: Total number of rows in `Employees`

57. **`SELECT * FROM Projects ORDER BY budget ASC;`**: Retrieves all projects, ordered by budget in ascending order.

- **Minimum Tuples**: 0 (if `Projects` table is empty)
- **Maximum Tuples**: Total number of rows in `Projects`

58. **`SELECT * FROM Departments ORDER BY dept_name;`**: Retrieves all departments, ordered alphabetically by department name.

- **Minimum Tuples**: 0 (if `Departments` table is empty)
- **Maximum Tuples**: Total number of rows in `Departments`

59. **`SELECT * FROM Works_On ORDER BY emp_id, proj_id;`**: Retrieves all work assignments, ordered first by employee ID and then by project ID.

- **Minimum Tuples**: 0 (if `Works_On` table is empty)
- **Maximum Tuples**: Total number of rows in `Works_On`

60. **`SELECT * FROM Employees ORDER BY dept_id, salary DESC;`**: Retrieves all employees, ordered first by department ID and then by salary in descending order.

- **Minimum Tuples**: 0 (if `Employees` table is empty)
- **Maximum Tuples**: Total number of rows in `Employees`

Queries with DISTINCT

61. **`SELECT DISTINCT dept_id FROM Employees;`**: Retrieves unique department IDs from the `Employees` table.

- **Minimum Tuples**: 0 (if `Employees` table is empty)
- **Maximum Tuples**: Number of unique `dept_id` in `Employees`

62. **`SELECT DISTINCT location FROM Departments;`**: Retrieves unique locations from the `Departments` table.

- **Minimum Tuples**: 0 (if `Departments` table is empty)
- **Maximum Tuples**: Number of unique `location` in `Departments`

63. **`SELECT DISTINCT proj_id FROM Works_On;`**: Retrieves unique project IDs from the `Works_On` table.

- **Minimum Tuples**: 0 (if `Works_On` table is empty)
- **Maximum Tuples**: Number of unique `proj_id` in `Works_On`

64. **`SELECT DISTINCT emp_id FROM Salaries;`**: Retrieves unique employee IDs from the `Salaries` table.
- **Minimum Tuples**: 0 (if `Salaries` table is empty)
- **Maximum Tuples**: Number of unique `emp_id` in `Salaries`

65. **`SELECT DISTINCT dept_name FROM Departments;`**: Retrieves unique department names from the `Departments` table.
- **Minimum Tuples**: 0 (if `Departments` table is empty)
- **Maximum Tuples**: Number of unique `dept_name` in `Departments`

Queries with IN and NOT IN

66. **`SELECT * FROM Employees WHERE dept_id IN (SELECT dept_id FROM Departments WHERE location = 'New York');`**: Retrieves all employees in departments located in New York.
- **Minimum Tuples**: 0 (if no employees are in New York departments)
- **Maximum Tuples**: Total number of rows in `Employees`

67. **`SELECT * FROM Projects WHERE dept_id NOT IN (SELECT dept_id FROM Departments WHERE location = 'San Francisco');`**: Retrieves all projects not associated with departments in San Francisco.
- **Minimum Tuples**: 0 (if all projects are associated with San Francisco departments)
- **Maximum Tuples**: Total number of rows in `Projects`

68. **`SELECT * FROM Employees WHERE emp_id IN (SELECT emp_id FROM Works_On WHERE proj_id = 'P123');`**: Retrieves all employees who work on project `P123`.
- **Minimum Tuples**: 0 (if no employees work on `P123`)
- **Maximum Tuples**: Total number of rows in `Employees`

69. **`SELECT * FROM Departments WHERE dept_id NOT IN (SELECT dept_id FROM Projects);`**: Retrieves all departments that have no associated projects.
- **Minimum Tuples**: 0 (if every department has at least one project)
- **Maximum Tuples**: Total number of rows in `Departments`

70. **`SELECT * FROM Employees WHERE emp_id IN (SELECT emp_id FROM Salaries WHERE annual_salary > 100000);`**: Retrieves all employees who have an annual salary greater than 100,000.
- **Minimum Tuples**: 0 (if no employees have an annual salary over 100,000)
- **Maximum Tuples**: Total number of rows in `Employees`

Queries with BETWEEN

71. **`SELECT * FROM Employees WHERE salary BETWEEN 50000 AND 100000;`**: Retrieves all employees with salaries between 50,000 and 100,000.
- **Minimum Tuples**: 0 (if no employees have a salary in this range)
- **Maximum Tuples**: Total number of rows in `Employees`

72. **`SELECT * FROM Projects WHERE budget BETWEEN 100000 AND 500000;`**: Retrieves all projects with a budget between 100,000 and 500,000.
- **Minimum Tuples**: 0 (if no projects have a budget in this range)
- **Maximum Tuples**: Total number of rows in `Projects`

73. **`SELECT * FROM Departments WHERE dept_id BETWEEN 1 AND 10;`**: Retrieves all departments with IDs between 1 and 10.
- **Minimum Tuples**: 0 (if no department IDs fall in this range)
- **Maximum Tuples**: Total number of rows in `Departments`

74. **`SELECT * FROM Works_On WHERE proj_id BETWEEN 'P100' AND 'P200';`**: Retrieves all work assignments for projects with IDs between `P100` and `P200`.

- **Minimum Tuples**: 0 (if no project IDs fall in this range)
- **Maximum Tuples**: Total number of rows in `Works_On`

75. **`SELECT * FROM Salaries WHERE annual_salary BETWEEN 60000 AND 120000;`**: Retrieves all salary records where the annual salary is between 60,000 and 120,000.

- **Minimum Tuples**: 0 (if no salary records fall in this range)
- **Maximum Tuples**: Total number of rows in `Salaries`

Queries with LIKE

76. **`SELECT * FROM Employees WHERE name LIKE 'A%';`**: Retrieves all employees whose names start with the letter 'A'.

- **Minimum Tuples**: 0 (if no employee names start with 'A')
- **Maximum Tuples**: Total number of rows in `Employees`

77. **`SELECT * FROM Projects WHERE proj_name LIKE '%Research%';`**: Retrieves all projects whose names contain the word 'Research'.

- **Minimum Tuples**: 0 (if no project names contain 'Research')
- **Maximum Tuples**: Total number of rows in `Projects`

78. **`SELECT * FROM Departments WHERE location LIKE 'New%';`**: Retrieves all departments located in places starting with 'New' (e.g., New York, New Delhi).

- **Minimum Tuples**: 0 (if no department locations start with 'New')
- **Maximum Tuples**: Total number of rows in `Departments`

79. **`SELECT * FROM Employees WHERE emp_id LIKE '_23';`**: Retrieves all employees with IDs ending in '23'.

- **Minimum Tuples**: 0 (if no employee IDs end in '23')
- **Maximum Tuples**: Total number of rows in `Employees`

80. **`SELECT * FROM Projects WHERE proj_name LIKE 'Project%';`**: Retrieves all projects whose names start with 'Project'.

- **Minimum Tuples**: 0 (if no project names start with 'Project')
- **Maximum Tuples**: Total number of rows in `Projects`

Queries with CASE

81. **`SELECT name, salary, CASE WHEN salary > 100000 THEN 'High' WHEN salary BETWEEN 50000 AND 100000 THEN 'Medium' ELSE 'Low' END AS salary_range FROM Employees;`**: Categorizes employees' salaries into 'High', 'Medium', or 'Low' ranges.

- **Minimum Tuples**: 0 (if `Employees` table is empty)
- **Maximum Tuples**: Total number of rows in `Employees`

82. **`SELECT proj_name, budget, CASE WHEN budget > 500000 THEN 'Large' WHEN budget BETWEEN 100000 AND 500000 THEN 'Medium' ELSE 'Small' END AS budget_category FROM Projects;`**: Categorizes projects' budgets into 'Large', 'Medium', or 'Small'.

- **Minimum Tuples**: 0 (if `Projects` table is empty)
- **Maximum Tuples**: Total number of rows in `Projects`

83. **`SELECT dept_name, location, CASE WHEN location = 'New York' THEN 'East Coast' WHEN location = 'San Francisco' THEN 'West Coast' ELSE 'Other' END AS region FROM Departments;`**: Categorizes departments based on their location into regions.

- **Minimum Tuples**: 0 (if `Departments` table is empty)
- **Maximum Tuples**: Total number of rows in `Departments`

84. **SELECT emp_id, proj_id, CASE WHEN proj_id LIKE 'P1%' THEN 'Phase 1' WHEN proj_id LIKE 'P2%' THEN 'Phase 2' ELSE 'Other' END AS project_phase FROM Works_On;**: Categorizes work assignments based on the project phase.

- **Minimum Tuples**: 0 (if `Works_On` table is empty)
- **Maximum Tuples**: Total number of rows in `Works_On`

85. **SELECT emp_id, CASE WHEN salary > 100000 THEN salary * 0.1 ELSE salary * 0.05 END AS bonus FROM Employees;**: Calculates a bonus for each employee based on their salary.

- **Minimum Tuples**: 0 (if `Employees` table is empty)
- **Maximum Tuples**: Total number of rows in `Employees`

Queries with JOIN (advanced)

86. **SELECT e.name, d.dept_name FROM Employees e JOIN Departments d ON e.dept_id = d.dept_id WHERE d.location = 'New York';**: Retrieves the names of employees working in departments located in New York.

- **Minimum Tuples**: 0 (if no employees work in New York departments)
- **Maximum Tuples**: Total number of rows in `Employees`

87. **SELECT p.proj_name, e.name FROM Projects p JOIN Works_On w ON p.proj_id = w.proj_id JOIN Employees e ON w.emp_id = e.emp_id WHERE p.budget > 100000;**: Retrieves the names of employees working on projects with a budget greater than 100,000.

- **Minimum Tuples**: 0 (if no employees work on high-budget projects)
- **Maximum Tuples**: Total number of rows in `Works_On`

88. **SELECT d.dept_name, COUNT(e.emp_id) FROM Departments d LEFT JOIN Employees e ON d.dept_id = e.dept_id GROUP BY d.dept_name;**: Counts the number of employees per department, including departments with no employees.

- **Minimum Tuples**: 0 (if `Departments` table is empty)
- **Maximum Tuples**: Number of unique `dept_name` in `Departments`

89. **SELECT p.proj_name, COUNT(w.emp_id) FROM Projects p LEFT JOIN Works_On w ON p.proj_id = w.proj_id GROUP BY p.proj_name;**: Counts the number of employees working on each project, including projects with no employees.

- **Minimum Tuples**: 0 (if `Projects` table is empty)
- **Maximum Tuples**: Number of unique `proj_name` in `Projects`

90. **SELECT e.name, s.annual_salary FROM Employees e RIGHT JOIN Salaries s ON e.emp_id = s.emp_id;**: Retrieves employee names along with their salaries, including salary records without matching employees.

- **Minimum Tuples**: 0 (if `Salaries` table is empty)
- **Maximum Tuples**: Total number of rows in `Salaries`

Queries with Self-JOIN

91. **SELECT e1.name, e2.name AS manager FROM Employees e1 JOIN Employees e2 ON e1.manager_id = e2.emp_id;**: Retrieves employee names along with their manager's name.

- **Minimum Tuples**: 0 (if no employees have managers)
- **Maximum Tuples**: Total number of rows in `Employees`

92. **`SELECT e1.name, e2.name AS colleague FROM Employees e1 JOIN Employees e2 ON e1.dept_id = e2.dept_id WHERE e1.emp_id <> e2.emp_id;`**: Retrieves pairs of employees who work in the same department.

- **Minimum Tuples**: 0 (if no employees work in the same department)
- **Maximum Tuples**: Number of pairs in `Employees` with matching `dept_id` but different `emp_id`

93. **`SELECT e1.name, e2.name AS teammate FROM Works_On w1 JOIN Works_On w2 ON w1.proj_id = w2.proj_id JOIN Employees e1 ON w1.emp_id = e1.emp_id JOIN Employees e2 ON w2.emp_id = e2.emp_id WHERE e1.emp_id <> e2.emp_id;`**: Retrieves pairs of employees who work on the same project.

- **Minimum Tuples**: 0 (if no employees work on the same project)
- **Maximum Tuples**: Number of pairs in `Works_On` with matching `proj_id` but different `emp_id`

94. **`SELECT e1.name, e2.name AS senior FROM Employees e1 JOIN Employees e2 ON e1.manager_id = e2.emp_id WHERE e1.salary < e2.salary;`**: Retrieves employees who earn less than their managers.

- **Minimum Tuples**: 0 (if no employees earn less than their managers)
- **Maximum Tuples**: Total number of rows in `Employees`

95. **`SELECT e1.name, e2.name AS junior FROM Employees e1 JOIN Employees e2 ON e1.manager_id = e2.emp_id WHERE e1.salary > e2.salary;`**: Retrieves employees who earn more than their managers.

- **Minimum Tuples**: 0 (if no employees earn more than their managers)
- **Maximum Tuples**: Total number of rows in `Employees`

Complex Queries with Subqueries, Joins, and Aggregates

96. **`SELECT dept_name, (SELECT AVG(salary) FROM Employees WHERE dept_id = d.dept_id) AS avg_salary FROM Departments d;`**: Retrieves department names along with the average salary of their employees.

- **Minimum Tuples**: 0 (if `Departments` table is empty)
- **Maximum Tuples**: Number of unique `dept_name` in `Departments`

97. **`SELECT proj_name, (SELECT COUNT(*) FROM Works_On WHERE proj_id = p.proj_id) AS num_employees FROM Projects p;`**: Retrieves project names along with the number of employees working on them.

- **Minimum Tuples**: 0 (if `Projects` table is empty)
- **Maximum Tuples**: Number of unique `proj_name` in `Projects`

98. **`SELECT name, (SELECT COUNT(*) FROM Works_On WHERE emp_id = e.emp_id) AS num_projects FROM Employees e WHERE EXISTS (SELECT 1 FROM Works_On WHERE emp_id = e.emp_id);`**: Retrieves employee names along with the number of projects they are working on, for employees working on at least one project.

- **Minimum Tuples**: 0 (if no employees work on any projects)
- **Maximum Tuples**: Total number of rows in `Employees`

99. **`SELECT dept_name, (SELECT COUNT(*) FROM Employees WHERE dept_id = d.dept_id) AS num_employees FROM Departments d WHERE EXISTS (SELECT 1 FROM Employees WHERE dept_id =`**

`d.dept_id);`: Retrieves department names along with the number of employees, for departments that have employees.

- **Minimum Tuples**: 0 (if no departments have employees)
- **Maximum Tuples**: Number of unique `dept_name` in `Departments`

100. **`SELECT proj_name, (SELECT SUM(salary) FROM Employees e JOIN Works_On w ON e.emp_id = w.emp_id WHERE w.proj_id = p.proj_id) AS total_salary FROM Projects p;`**: Retrieves project names along with the total salary of employees working on each project.

- **Minimum Tuples**: 0 (if `Projects` table is empty)
- **Maximum Tuples**: Number of unique `proj_name` in `Projects`