```
create database manager;
use manager;

create schema employee;
use schema employee;

create or replace table departments(
 department_id int primary key autoincrement,
 department_name varchar(100)unique not null
);
INSERT INTO departments (department_name) VALUES
('Human Resources'),
('Finance'),
('Information Technology'),
('Marketing'),
('Sales'),
('Operations'),
('Customer Support'),
('Research and Development'),
('Legal'),
('Administration');

 select*from departments;
```

……

```
create or replace table employees(
 employee_id int  autoincrement primary key,
 first_name varchar(50) not null,
 last_name varchar(50) not null,
 department_id int foreign key references departments(department_id),
 hire_date date not null
 );

insert into employees (first_name, last_name, department_id, hire_date) values('Alice',
'Johnson', 1, '2022-11-15'),
('Bob', 'Smith', 2, '2023-02-10'),
('Charlie', 'Brown', 3, '2023-06-01'),
('Diana', 'Miller', 4, '2021-09-20'),
('Ethan', 'Wilson', 2, '2023-03-05'),
('Fiona', 'Clark', 3, '2023-07-12'),
('George', 'Lopez', 1, '2022-12-30'),
('Hannah', 'White', 4, '2023-04-22'),
('Ian', 'Taylor', 3, '2023-08-14'),
('Julia', 'Davis', 2, '2020-05-18');


Select *from employees;
```

……

```
create table projects(
 project_id int primary key autoincrement,
 project_name varchar(100) not null,
 start_date date,
 end_date date,
 department_id int foreign key references departments(department_id)
 );

INSERT INTO projects (project_name, start_date, end_date, department_id) VALUES
('Employee Onboarding Program', '2023-01-10', '2023-03-30', 1), -- HR
('Annual Budget Planning', '2023-02-01', '2023-04-15', 2),      -- Finance
('IT Infrastructure Upgrade', '2023-03-01', '2023-07-31', 3),   -- IT
('Social Media Marketing Campaign', '2023-04-01', '2023-06-30', 4), -- Marketing
('Global Sales Strategy', '2023-05-01', '2023-08-15', 5),       -- Sales
('Logistics Optimization', '2023-06-01', '2023-09-30', 6),      -- Operations
('Customer Service Chatbot', '2023-07-10', '2023-10-20', 7),    -- Customer Support
('AI Product Research', '2023-08-01', '2024-01-15', 8),         -- R&D
('Regulatory Compliance Audit', '2023-09-01', '2023-11-30', 9), -- Legal
('Head Office Renovation', '2023-10-01', '2023-12-31', 10);     -- Administration

select* from projects;
```

……

```
create table employeeprojects(
 employee_id int foreign key references employees (employee_id),
 project_id int foreign key references projects(project_id),
 assigned_date date not null
);

insert into employeeprojects(employee_id,project_id,assigned_date)values (1, 101,
'2024-01-15'),
(2, 101, '2024-01-20'),
(3, 102, '2024-02-05'),
(1, 103, '2024-02-10'),
(4, 103, '2024-02-15'),
(5, 104, '2024-03-01'),
(2, 104, '2024-03-05'),
(3, 105, '2024-03-10'),
(4, 106, '2024-04-01'),
(5, 106, '2024-04-05');

select*from employeeprojects;
```

……

```
create or replace table salaries(
 salary_id int primary key autoincrement,
 employee_id int foreign key references employees(employee_id),
 salary_amount decimal(10,2),
 effective_date date not null
```

```
);

insert into salaries(employee_id, salary_amount, effective_date) VALUES
(1, 55000.00, '2023-01-15'),
(2, 60000.00, '2023-02-01'),
(3, 45000.00, '2023-03-01'),
(4, 70000.00, '2023-04-10'),
(5, 52000.00, '2023-05-01'),
(1, 58000.00, '2024-01-01'),  -- salary update for employee 1
(2, 63000.00, '2024-02-01'),  -- salary update for employee 2
(3, 47000.00, '2024-03-01'),
(4, 72000.00, '2024-04-15'),
(5, 54000.00, '2024-05-01');

select*from salaries;
```

……

2. Show all departments sorted alphabetically by department_name

```
select*from departments
order by department_name;
```

3. List all departments whose names contain the word "Sales"

```
select * from departments
where department_name like 'Sales';
```

2. Retrieve all salaries greater than 50,000 from the salaries table, ordered by salary_amount

```
select*from salaries
where salary_amount>50000
order by salary_amount;
```

5. Show all employees who belong to the IT department (use subquery or JOINe as needed)

```
select e.first_name, e.last_name
from employees e
join departments d
on e.department_id=d.department_id
where d.department_name = 'Information Technology';
```

3. Display the salaries table sorted by effective_date and then by salary_amount descending

```
select*from salaries
order by effective_date,salary_amount desc;
```

```
select distinct(salary_amount) from salaries
order by salary_amount desc
limit 1 offset 4;
```

1. Write and execute the SQL statement(s) to modify the employees table by adding the following columns:
○ gender: ENUM with values 'Male', 'Female', 'Other'.

```
select*from employees;
alter table employees add column gender varchar(100);
update employees set gender='male'
where employee_id in (1,2);

update employees set gender='female'
where employee_id in (3,9);

update employees set gender='other'
where employee_id in (4,7,8);
```

○ manager_id: INTEGER that references another employee (emp_id), nullable for top-level managers.

```
alter table employees add column Manager_id int foreign key references
employees(employee_id);

select*from employees;
```

11.1 Aggregate Functions Practice
Tasks 1-5: Basic Aggregate Operations

1. Count total employees: Write a query to count the total number of employees in the company

```
select count(*)employee_name from employees;
```

2. Calculate total salary expenditure: Find the total salary amount paid to all employees

```
select sum(salary_amount)from salaries;
```

3. Determine average salary: Get the average salary of employees in the salaries table

```
select avg(salary_amount)from salaries;
```

4. Find maximum salary: Identify the highest salary amount ever paid

```
select max(salary_amount) from salaries;
```

5. Identify earliest hire date: Find the earliest hire_date from the employees table

```
select min(hire_date)from employees;
```

11.2 Scalar Functions Practice
Tasks 6-10: String and Date Function Applications

6. Uppercase transformation: Display all employee first names in uppercase

```
select upper(first_name)as firstname
from employees;
```

7. Lowercase conversion: Display all project names in lowercase

```
select lower(project_name)as projectname
from projects;
```

9. String length analysis: Retrieve the length of each employees first name

```
select len(first_name)as named
from employees;
```

10.Name concatenation: Display full names (first + last) of all employees in one column as "Full Name"

```
select concat(first_name,last_name)as fullname
from employees;

select first_name,'',last_name as fullname
from employees;

select first_name || last_name as fullname
from employees;
```

11.3 JOIN Operations Practice
Tasks 11-13: INNER JOIN Applications
11.Employee-Department listing: List employees along with their department names

```
select concat(e.first_name,e.last_name) as fullname,d.department_name
from departments as d join employees as e
on d.department_id = e.department_id;
```

12.Project-Department mapping: Show project names and the departments they belong to

```
select p.project_name,d.department_name
from projects as p join departments as d
```

```
on d.department_id = p.department_id;
```

13.Employee project assignments: List each employee along with their project assignments and assigned dates

```
select concat(e.first_name,e.last_name) as fullname,e1.assigned_date
from employeeprojects as e1 join employees as e
on e.employee_id = e1.employee_id;
```

Tasks 14-15: LEFT JOIN Applications
14.Complete employee listing: List all employees and their department names (include those not assigned to departments)

```
select concat(e.first_name,e.last_name) as fullname,d.department_name
from departments as d left join employees as e
on d.department_id = e.department_id;
```

15.Complete project listing: List all projects and their assigned employees (include projects not assigned to any employee)

```
select concat(e.first_name,e.last_name) as fullname,p.project_name
from projects as p left join employees as e
on e.department_id = p.department_id;
```

Tasks 16-17: RIGHT JOIN Applications
16.Complete department listing: Show all departments and employees in them(include departments without employees)

```
select concat(e.first_name,e.last_name) as fullname,d.department_name
from departments as d right join employees as e
on d.department_id = e.department_id;
```

17.Employee project coverage: List all employees and the projects they are assigned to, showing all employees (even if not assigned)

```
select concat(e.first_name,e.last_name) as fullname,p.project_name
from projects as p right join employees as e
on e.department_id = p.department_id;
```

Tasks 18-19: FULL OUTER JOIN Simulation
18.Department-Employee union: Write a query to list all departments and employees (including unmatched ones from both tables)

```
select concat(e.first_name,e.last_name) as employeename,d.department_name
from departments as d full outer join employees as e
on d.department_id = e.department_id;
```

19.Employee-Salary comprehensive view: Display all employees and their

salaries, including employees without salary records and salaries not linked to any employee

```
select concat(e.first_name,e.last_name) as employeename,s.salary_amount
from employees as e full outer join salaries as s
on e.employee_id = s.employee_id
```

11.4 Advanced Grouping Practice
Tasks 22-28: GROUP BY and HAVING Applications
22.Department employee count: Display each departments name along with the number of employees working in that department

```
select count(first_name ,'', last_name) as empname,d.department_name
from employees as e
join departments as d
on d.department_id = e.department_id
group by d.department_name;
```

23.Department salary analysis: Show the average salary of employees grouped by their department names

```
select avg(salary_amount)as avg_salary,d.department_name
from departments as d
join employees as e join salaries as s
on d.department_id = s.department_id
group by d.department_name;
```

28.Project employee assignment: Show each project name along with the total

number of employees assigned to that project

```
select d.department_name,count(project_id)as project_count
from departments d join projects p
on d.department_id = p.department_id
group by d.department_name;
```

24.Department project count: List each department name and how many projects are handled under it

```
select p.project_name,count(employee_id)as number_emp
from employees as e join projects as p
on e.department_id = p.department_id
group by p.project_name;
```

25.Department filtering: Show departments that have more than two employeesusing the HAVING clause

```
select e.employee_id,d.department_name
from employees as e
join department as d
```

```
on d.department_id = e.department_id
group by e.employee_id,d.department_name
having count(e.employee_id) > 2;
```

26.Employee salary frequency: List all employees who have received more than one salary entry in the salaries table

```
select e.employee_id,s.salary_id
from employees as e
join salaries as s
on e.employee_id = s.employee_id
group by e.employee_id,s.salary_id
having sum(salary_amount)>1;
```

27.Employee total compensation: Calculate and display the total salary received by each employee across all their salary records

```
select sum(salary_amount)as salary_amount,e.employee_id
from salaries as s join employees as e
on e.employee_id = s.employee_id
group by e.employee_id;
```

15.1 Subquery Practice
Tasks 1-3: Advanced Data Filtering
1. Maximum salary identification: List all employees who have the highest salary

```
select e.employee_id,s.salary_amount
from employees as e join salaries as s
on e.employee_id = s.employee_id
where salary_amount = (select max(salary_amount)from salaries);
```

```
select employee_id,salary_amount from salaries
where salary_amount = (select max(salary_amount)from salaries);
```

2.Department comparison analysis: Show departments with more employees than the average number of employees per department

```
select departments,avg(employee) from department
```

3. Above-average salary analysis: Find employees who earn more than the average salary

```
select employee_id,salary_amount from salaries
where salary_amount>(select avg(salary_amount)from salaries);
```

15.2 View Creation Practice
Tasks 4-5: Virtual Table Development
4. Active employee view: Create a view active_employees showing emp_id, first_name, and department_name

```
create or replace view active_employees as
select e.employee_id,e.first_name,d.department_name
from departments as d join employees as e
on d.department_id = e.department_id;

select * from active_employees views;
```

5. Project summary view: Create a view project_summary showing project_name and number of employees assigned

```
create or replace view project_summary as
select p.project_name,count(e.employee_id) as emp_id
from projects as p join employees as e
on e.department_id = p.department_id
group by p.project_name;

select * from project_summary view;
```

1.Create a view that shows all employees with salary greater than 50000.

```
create view sales_man as
select employee_id,salary_amount
from salaries
where salary_amount>50000;

select *from sales_man view;
```

3.Create a view that displays only employees from the HR department.

```
create view hr_names as
select e.employee_id,d.department_name
from departments as d join employees as e
on d.department_id = e.department_id
where department_name ='Human Resources';

select*from hr_names view;
```

4.Create a view that shows employees along with their department names (using a join).

```
create or replace view department_name as
select e.employee_id,d.department_name
from departments as d
join employees as e
on d.department_id = e.employee_id

select * from department_name view;
```

7.Create a view that shows departments having more than 5 employees.

```
create view emp_dept as
select e.employee_id,e.department_id
from employees e
where employee_id > 5;

select * from emp_dept view
```

1.Find employees who earn more than the average salary.

```
select *from salaries;

select employee_id,salary_amount
from salaries as s
where salary_amount > (select avg(salary_amount) from salaries
where salary_amount >70000);

select employe_id,salary_amount
from salaries as s
where salary_amount = (select salary_amount from employees where)
```

1.Find employees who earn more than the average salary.

```
select employee_id,salary_amount
from salaries as s
where salary_amount>(select avg(salary_amount)from salaries);
```

2. Department comparison analysis: Show departments with more employees than the average number of employees per department

```
select d.department_name,count(employee_id)as emp_id
from departments as d join employees as e
on d.department_id = e.department_id
```

```
group by d.department_name
having avg(employee_id) > (select avg(employee_id)from employees);
```

3.Find the employees who earn more than the highest salary in department 60k.

```
select first_name || last_name as emp_name,e.department_id,s.salary_amount
from salaries as s join employees as e
on e.employee_id = s.employee_id
where salary_amount > 60000;
```

4.Above-average salary analysis: Find employees who earn more than the average salary

```
select employee_id,salary_amount
from salaries as s
where salary_amount > (select avg(salary_amount)from salaries);
```

1. Create a unique index on department_name in the departments table

```
create or replace table departments(
 department_id int,
 department_name varchar(100)unique
)
 cluster by (department_name);

 select*from departments
```

2. Create an index on last_name in the employees table for optimized surname searches

```
create or replace table employees(
  first_name varchar(100),
  last_name varchar(100)
)
 cluster by(last_name);
```

3.Create a composite index on (department_id, hire_date) in the employees table

```
create or replace table employees(
 department_id int,
 hire_date date
 )
 cluster by (department_id,hire_date);
select*from employees
```

4. Create an index on salary_amount in the salaries table for salary range queries

```
create or replace table salaries(
  salary_id int,
  salary_amount int
)
cluster by(salary_amount);
```

5. Drop the index created on last_name from the employees tablengrhmef

```
alter table employees drop clustering key;

alter table employees cluster by (department_name);
```

1.Write a query to display each employee's ID, department, salary, and the average salary of their department using PARTITION BY.

```
select e.employee_id,e.department_id,s.salary_amount,
avg(s.salary_amount)
over(partition by e.department_id)
as dept_id
from salaries as s
join employees as e
on e.employee_id = s.employee_id
order by department_id;

select*from salaries;
```

Tasks 20-21: SELF JOIN and CROSS JOIN

20.Employee hire date comparison: Show pairs of employees where one was hired before the other.

```
select employee_id,project_name
from employees
cross join projects
order by employee_id;
```

related subqueries:
1.From the employees table, find the employees who earn more than the average salary of their department.

```
select e.employee_id,e.department_id,e.salary_amount
from employees as e
where e.salary_amount > (select avg(s.salary_amount)as salaryed from employees s
where s.department_id=e.department_id)
```

2.Find all employees whose salary is greater than the average salary of all employees.

```
select salary_amount,employee_id
from salaries as salary_name
where salary_amount>(select avg(salary_amount)as salary_amount from salaries);
```

```
select*from salaries
```

1.Find departments that have at least one employee.

```
select department_id,department_name
from departments d
where exists(
  select *
  from employees e
  where d.department_id=e.department_id
);
```

2.Departments with no employees:

```
  select department_name
  from departments d
  where not exists(
   select *
   from employees e
   where d.department_id=e.department_id
);
```

3.Find employees who were hired after January 1, 2023, and whose department exists.

```
select department_name
from departments as d
where exists(
  select*
  from employees as e
  where e.department_id = d.department_id
  and e.hire_date > '2023-01-01'
);
```

4.Find employees who work in departments with names containing the word 'Finance'.

```
select department_name
from departments d
```

```
where exists(
select 1
from employees as e
where d.department_id = e.department_id
and department_name = 'Finance'
);
```

5.List departments that have more than 2 employees (use EXISTS with a subquery).

```
select employee_id
from employees e
where exists(
select *
from departments d
where d.department_id = e.department_id
and department_id>2
);
```

6.Find departments where at least one employee was hired before 2022

```
select department_id,department_name
from departments as d
where exists(
select*
from employees as e
where d.department_id= e.department_id
and e.hire_date <'2022-01-01'
);
```

subqueries;

1.Find employees who work in the same department as 'Alice Johnson'.

```
select first_name || last_name
from employees as e
where department_id = (select department_id from employees as e
where first_name='Alice' and last_name= 'Johnson');
```

2.Find the employee(s) with the earliest hire date.

```
select first_name || last_name
from employees as e
where hire_date=(select min(hire_date)from employees as e);
```

3.Find departments where the average hire year is after 2022.

```
select d.department_id,avg(year(e.hire_date))as avg
from departments as d
join employees as e
on d.department_id = e.department_id
group by d.department_id
having avg(year(e.hire_date))>2022;
```

9.Show employees who have the earliest hire date in their department.

```
select e.hire_date,d.department_name
from employees as e
join departments as d
on d.department_id = e.department_id
where e.hire_date=(select min(e.hire_date)
from employees as e
where d.department_id = e.department_id)
```

1.Write a query to assign a unique row number to employees ordered by salary (highest first).

```
select employee_id,salary_amount,
row_number ()over (partition by employee_id
order by salary_amount desc)as salary_name
from salaries;
```

2.Rank employees' salaries within each department, showing ties if two employees have the same salary.

```
select e.employee_id,s.salary_amount,d.department_name,
rank () over(
partition by d.department_name
order by s.salary_amount desc) as salared
from salaries as s
join employees as e
on e.employee_id = s.employee_id
join departments as d
on d.department_id = e.department_id
```

2.Write a query to display each employee's salary along with the rank of their salary in the company.

```
select employee_id,salary_amount,
```

```
rank() over(
partition by employee_id
order by salary_amount)as salary_amount
from salaries;
```

3.Display employees with their salary and the dense rank of salary within each department.

```
select e.first_name || e.last_name,s.salary_amount,e.department_id,
rank ()over(
partition by salary_amount
order by department_id desc) as dept_id
from salaries as s
join employees as e
on e.employee_id = s.employee_id;
```

4.Find the running total of salaries ordered by employee hire date.

```
select s.salary_amount,e.hire_date,
sum(salary_amount) over(
partition by salary_amount
order by hire_date asc)as hire_date
from employees as e
join salaries as s
on e.employee_id = s.employee_id;
```

5.Display the average salary per department using a window function.

```
select s.salary_amount,e.department_id,e.employee_id,
avg(salary_amount) over(
partition by s.salary_amount
order by employee_id desc)as emp_id
on e.employee_id = s.employee_id;
```

6.Show each employee’s salary along with the next employee’s salary based on hire date

```
select e.first_name || e.last_name,s.salary_amount,e.hire_date,
lead(salary_amount)over (
order by hire_date desc)as hire_date
from employees as e
join salaries as s
on e.employee_id = s.employee_id

CREATE OR REPLACE PROCEDURE CalculateLateFees(member_id_input INT)
RETURNS FLOAT
LANGUAGE SQL
AS
$$
DECLARE
```

```
   total_late_fees FLOAT;
BEGIN
   SELECT COALESCE(
            SUM(DATEDIFF('day', due_date, CURRENT_DATE()) * 0.25),0)
     INTO total_late_fees
     FROM Loans
    WHERE member_id = member_id_input
      AND return_date IS NULL
      AND due_date < CURRENT_DATE();

   RETURN total_late_fees;
END;
$$;

CALL CalculateLateFees(4);
```

CTE:

1.Write a query using CTE to find employees who earn above the average salary.

```
WITH AvgSalary AS (
   SELECT AVG(salary_amount) AS avg_sal
   FROM employees
)
SELECT emp_id, name, salary
FROM employees, AvgSalary
WHERE employees.salary > AvgSalary.avg_sal;
```

1.Write a query using CTE to find employees who earn above the average salary.

```
WITH avg_salary AS (
   SELECT AVG(salary_amount) AS average_salary
   FROM salaries
)
SELECT e.employee_id, e.first_name || e.last_name, s.salary_amount
FROM employees e
join salaries s
on s.employee_id = e.employee_id,
avg_salary as a
WHERE s.salary_amount > a.average_salary;
```

2.Create a CTE that calculates the average salary per department.

```
with avg_salary as (
  select avg(salary_amount)as average_salary
  from salaries
)
```

```
select d.department_id,d.department_name
from departments d
join salaries s
on s.department_id = d.department_id
```