DDL

- creating database:

```
CREATE DATABASE DDL;
use DDL;
```

- 1. Create a Table

```
CREATE TABLE employees(
id int AUTO_INCREMENT PRIMARY KEY,
name varchar(100) NOT NULL,
designation varchar(50),
salary decimal(10,2) DEFAULT 5000
)
```

- 2. Add a Column:

```
ALTER TABLE `employees` ADD COLUMN joining_date Date;
```

- 3. Modify a Column:

```
ALTER TABLE `employees`
MODIFY COLUMN employees.joining_date float;
```

- 4. Drop a Column:

```
ALTER TABLE `employees` DROP COLUMN employees.joining_date;
```

- 5. Rename a Column:

```
ALTER TABLE `employees`
CHANGE employees.designation job_title varchar(50)
```

- 6. Rename A Table

```
RENAME TABLE `employees` to company_employees;
```

- 7. Drop a Table:

```
DROP TABLE `company_employees`;
```

- 8. Truncate a Table: <!--deleting all data--!>

```
TRUNCATE TABLE `table_name`;
```

DML

- 9. Insert three employees data:

```
INSERT INTO `employees`(employees.id,employees.name,employees.job_title,employees.salary)
VALUES
(3,"Alice Brown","Analyst",7000),
   (4,"Bob White","Tester",8000),
(5,"Charlie Black","Designer",6000);
```

- 10. Select All Rows:

```
SELECT * FROM `employees` WHERE 1;
```

- 11. Select Specific Columns:

```
SELECT employees.name,employees.designation FROM `employees` WHERE 1;
```

- 12. Select with Condition:

```
SELECT * FROM `employees` WHERE employees.salary > 8000;
```

- 13. Select with LIKE operator:

```
SELECT employees.name  FROM `employees` WHERE employees.name LIKE "j%";
```

- 14. Update Salary:

```
UPDATE `employees` SET employees.salary = 12000 WHERE employees.id = 1;
```

- 15. Update Multiple Columns:

```
UPDATE `employees` SET employees.salary  = 15000 and employees.designation = 'Senior Developer' WHERE employees.id = 2;
```

- 16. Delete a Row:

```
DELETE FROM `employees` WHERE employees.id = 5;
```

- 17. Delete Rows With a Condition

```
DELETE FROM `employees` WHERE employees.salary < 7000;
```

- 18. Select with ORDER BY:

```
SELECT * FROM `employees` ORDER BY employees.salary DESC;
```

DISTINCT , AND , OR , NOT , BETWEEN and LIKE

- 19. List unique designation in this employees table:

```
SELECT DISTINCT employees.designation,employees.salary FROM `employees` WHERE 1;
```

- 20. Find the unizue combinations of designations of designation and salary:

```
SELECT DISTINCT employees.designation and employees.salary FROM `employees` WHERE 1;
```

- 21. Cound the number of unizue designation;

```
SELECT COUNT(DISTINCT employees.designation) FROM `employees` WHERE 1;
```

- 22. Find employees with a salary greater than 5000 and a designation of 'Manager':

```
SELECT * FROM `employees` WHERE employees.salary > 5000 and employees.designation = 'Manager';
```

- 23. Retrieve employees whose name starts with 'A' and salary is between 5000 and 10000:

```
SELECT * FROM `employees` WHERE employees.name LIKE 'A%' and salary > 5000 and salary < 10000;
```

- 24. Find employees with a designation of 'Developer' and salary less than 20000:

```
SELECT * FROM `employees` WHERE employees.designation = 'Developer' AND employees.salary < 20000;
```

- 25. Find employees with a salary greater than 50000 or a designation of 'Intern':

```
SELECT * FROM `employees` WHERE employees.salary > 50000 OR employees.designation = 'Intern';
```

- 26. Retrieve employees whose name starts with 'J' or salary is exactly 5000:

```
SELECT * FROM `employees` WHERE employees.name like "J%" OR employees.salary = 5000;
```

- 27. List employees who are either 'Manager' or 'Team Lead':

```
SELECT * FROM `employees` WHERE employees.designation = 'Manager' OR employees.designation = 'Team Lead';
```

- 28. Find employees who do not have a salary equal to 5000:

```
SELECT * FROM `employees` WHERE employees.salary != 5000;
```

- 29. Retrieve emplo yees whose designation is not 'Developer':

```
SELECT * FROM `employees` WHERE `employees`.`designation` != 'Developer';
```

- 30. Find employees whose name does not contain the letter 'e':

```
SELECT * FROM `employees` WHERE `employees`.`name` NOT LIKE "%e%";
```

- 31. Find employees whose salary is between 10000 and 30000:

```
SELECT * FROM `employees` WHERE `employees`.`salary` BETWEEN 10000 AND 30000;
```

- 32. Retrieve employees with id values between 9 and 15:

```
SELECT * FROM `employees` WHERE `employees`.`id` BETWEEN 5 AND 15;
```

- 33. Find employees whose salary in not between 5000 and 20000:

```
SELECT * FROM `employees` WHERE employees.salary NOT BETWEEN 5000 AND 20000;
```

- 34. Find all employees whose names start with the letter A.

```
SELECT * FROM `employees` WHERE employees.name LIKE "A%";
```

- 35. Find all employees whose name end with the letter n.

```
SELECT * FROM `employees` WHERE employees.name LIKE "%n";
```

- 36. Find all employees whose names contain the substring John.

```
SELECT * FROM `employees` WHERE employees.name LIKE "%John%";
```

- 37. Find all employees whose name are exactly 5 characters long

```
SELECT * FROM `employees` WHERE LENGTH(`employees`.`name`) = 5;
```

- 38. Find all employees whose designations start with M and end with r

```
SELECT * FROM `employees` WHERE employees.designation LIKE "M%" AND employees.designation LIKE "%r";
```

- 39. Find all employees whose names have e as the second letter.

```
SELECT * FROM `employees` WHERE employees.name LIKE "_e%";
```

- 40. Find all employees whose names do not contain the letter z

```
SELECT * FROM `employees` WHERE employees.name NOT LIKE "%z%";
```

Aggregate Function

- 41. write a query to calculate the total salary of all employees in the employees table.

```
SELECT SUM(employees.salary) as total_salary FROM `employees` WHERE 1;
```

- 42. write a query to find the average salary of employees.

```
SELECT AVG(employees.salary) as avg_salary FROM `employees` WHERE 1;
```

- 43. write a query to count the total number of employees in the table.

```
SELECT COUNT(employees.id) as total_emp FROM `employees` WHERE 1;
```

- 44. write a query to find the highest salary among employees.

```
SELECT MAX(employees.salary) as max_salary FROM `employees` WHERE 1;
```

- 45. write a query to find the lowest salary in the employees table.

```
SELECT MIN(employees.salary) as min_salary FROM `employees` WHERE 1;
```

- 46. write a query to count the number of employees

```
SELECT COUNT(employees.id) as total_emp FROM `employees` WHERE 1;
```

GROUP BY , HAVING, ORDER BY

- 47. write a query to find the total salary per department.

```
SELECT SUM(employees.salary) as total_salary_per_dept FROM `employees` GROUP BY employees.designation;
```

- 48. Find the average salary of employees in the 'HR' department

```
SELECT AVG(employees.salary) as total_salary_per_dept FROM `employees` WHERE employees.designation='HR' GROUP BY employees.desi
```

- 49. List department with more than 1 employee and their average salary

```
SELECT employee.department , AVG(employee.salary) as avg_salary FROM `employee` GROUP BY employee.department HAVING COUNT(emplo
```

- 50. Order employees by salary in descending order.

```
SELECT * FROM `employee` ORDER BY employee.salary DESC;
```

- 51. Find the highest salary in each departemnt ```

```
SELECT * FROM `employee` GROUP BY employee.department ORDER BY employee.salary DESC LIMIT 1;
```

- 52. List all employees who have a salary greater than 5000 and order them by name.

```
SELECT * FROM `employee` WHERE employee.salary > 5000 ORDER BY employee.name;
```

- 53. Count the number of employees in each department and only show departemts with more than 1 employee.

```
SELECT COUNT(employee.employee_id), department FROM `employee` GROUP BY employee.department HAVING COUNT(employee.employee_id)
```

- 54. Find the employees who were hired after '2020-01-01' and order them by hire_date.

```
SELECT * FROM `employee` WHERE employee.hire_date > '2020-01-01' ORDER BY employee.hire_date;
```

- 55. Get the sum of salaries of employees hire in 2020 or later

```
SELECT SUM(employee.salary) as sum_salary FROM `employee` WHERE employee.hire_date > '2020-01-01';
```

- 56. List employees who earn less then 5000 and order them by salary in ascending order

```
SELECT * FROM `employee` WHERE employee.salary <5000 ORDER BY employee.salary ASC;
```

- 57. Find the department with the maximum average salary

```
SELECT department, AVG(salary) AS avg_salary FROM employee GROUP BY department ORDER BY avg_salary DESC LIMIT 1;
```

- 58. List the departments that have employees with a salary greater than 6000

```
SELECT department FROM employee WHERE employee.salary > 6000;
```

STORED PROCEDURE , VIEWS and TRIGGERS- 01

- creating table:

```
CREATE TABLE employees(
employee_id int AUTO_INCREMENT PRIMARY KEY,
name varchar(50),
department varchar(20),
salary int );

INSERT INTO employees(name, department , salary)VALUES
("Alice","HR",5000),
("Bob","IT",6000),
("Charlie","HR",5500)
```

```
CREATE TABLE departments(department_id int AUTO_INCREMENT PRIMARY KEY,department_name varchar(20));

INSERT INTO departments(department_name)
VALUES("HR"),("IT");
```

```
CREATE TABLE projects(project_id int AUTO_INCREMENT PRIMARY KEY, project_name varchar(30),department_id int);

INSERT INTO projects(project_name , department_id)
VALUES('Project A', 1),
   ('Project B', 2);
```

- 1. Creat a Stred Procedure to get employee salary by department

```
DELIMITER $$
CREATE PROCEDURE emp_salary_by_dep(department_name varchar(20))
BEGIN
    SELECT `employees`.`salary` FROM `employees`
  WHERE employees.department = department_name
END $$
DELIMITER ;
```

- 2. Create a Stored Procedure to Update Employee Salary.

```
DELIMITER $$
CREATE PROCEDURE update_emp_salary(emp_id int ,salary int)
BEGIN
    UPDATE `employees` SET employees.salary = salary WHERE employees.employee_id = emp_id;
END $$
DELIMITER ;
```

- 3. Create a Stored Procedure to Inset a new Employee

```
DELIMITER $$
CREATE PROCEDURE insert_emp(name varchar(50) , department varchar(20) , salary int)
BEGIN
    INSERT INTO `employees`(employees.name , employees.department , employees.salary)
  VALUES(name , department , salary);
END $$
DELIMITER ;
```

- 4. Create a Stored Procedure to Get Employees with Salary Greater Than a Certain Amount

```
DELIMITER $$
CREATE PROCEDURE emp_salarygratter(salary int)
BEGIN
     SELECT * FROM `employees` WHERE employees.salary > salary;
END $$
DELIMITER ;
```

- 5. Create a Stored Precedure to Delete Employee by ID

```
DELIMITER $$
CREATE PROCEDURE emp_delete(ID int)
BEGIN
    DELETE FROM `employees` WHERE employees.employee_id = ID;
END $$
DELIMITER ;
```

- 6. Create a View to Show Employees and Their Departments

```
CREATE VIEW emp_and_dept AS
SELECT employees.employee_id , employees.name FROM `employees`;
```

- 7. Create a View to Show Employees with Salary Above 6000

```
CREATE VIEW emp_salay AS
SELECT employees.employee_id , employees.name, employees.salary
FROM `employees` WHERE employees.salary > 6000;
```

- 8. Create a View to Show Projects with their Department Names

```
CREATE VIEW project_dept AS
SELECT projects.project_name , departments.department_name FROM `projects`
JOIN `departments` ON projects.department_id = departments.department_id;
```

- 9. Create a View to Get Employee and Project Details

```
create view emp_project as
SELECT employees.employee_id,employees.name,`departments`.`department_name`,employees.salary , `projects`.`project_name` FROM `

JOIN `departments` ON `employees`.`department` = `departments`.`department_name`
JOIN `projects` ON `departments`.`department_id` = `projects`.`department_id`;
```

- 10. create a View to Show Total Salary of Employee in Each Department

```
CREATE VIEW IF NOT EXISTS sum_salary_dept AS
SELECT SUM(`employees`.`salary`) FROM `employees`
GROUP BY `employees`.`department`;
```

- 11. Create a Trigger to Automatically Update Employees in Each Department
- creating the table:

```

```

CREATE TABLE salary_history(employee_id int, old_salary int , new_salary int)

```
 ```sql
DELIMITER $$
CREATE TRIGGER update_salary
AFTER UPDATE ON `employees`
FOR EACH ROW
BEGIN
          INSERT INTO `salary_history`(salary_history.employee_id ,                                                    salary_history.old_s
   VALUES(NEW.`employee_id`,OLD.salary , NEW.salary);
END $$
DELIMITER ;
```

- 12. Create a Trigger to Prevent Deleting Employees with Salary Above 7000

```
DELIMITER $$
CREATE TRIGGER prevent_del
BEFORE DELETE ON `employees` FOR EACH ROW
BEGIN
IF OLD.salary > 7000 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = "CAN'T DELETE , because salary > 7000" ;
END IF ;
END $$
DELIMITER ;
```

- 13. Create a Trigger to Automatically Update Department's Average Salary After Salary Change
- creating new table avg_salary

```
CREATE TABLE avg_salary_dept(department_id int AUTO_INCREMENT PRIMARY KEY,
department_name varchar(20),
avg_salary int )
```

Not correct.
```

```
DELIMITER $$
CREATE TRIGGER avg_dept_salary
AFTER UPDATE ON `employees`
BEGIN
INSERT IF NOT EXISTS INTO avg_salary_dept(
    avg_salary_dept.department_name,
    avg_salary_dept
)
 VALUES(
NEW.department,(SELECT AVG(`employees`.`salary`)
FROM `employees`
WHERE `employees`.`department` = NEW.department) ;
```

END $$ DELIMITER;

- 14. Create a Trigger to Insert a Record in the Audit Table After Employee Insert
- creating table `audit`

```
CREATE TABLE audit(employee_id int , action varchar(20) , action_date date DEFAULT CURRENT_DATE)
```

```
DELIMITER $$
CREATE TRIGGER audit_trigger
AFTER INSERT ON `employees`
FOR EACH ROW
BEGIN
   INSERT INTO `audit`(audit.employee_id,audit.action)
  VALUES(NEW.employee_id,"insert");
END $$
DELIMITER ;
```

- 15. Creating a Trigger to Update Project's Department When Department ID Changes

Stored Procedure, Views and Triggers - 02