QUERIES

1) Find student_id, name, and course name of Students who take 2 course : Database and FrontEnd;

```
SELECT Student.student_id, Student.name, Course.course_id, Course.name
FROM Student, Course, Enroll
WHERE Student.student_id = Enroll.student_id AND Enroll.course_id = Course.course_id
AND (COURSE.name = 'FrontEnd' OR Course.name = 'Database');
```

2) Find the id and name of Students who live in Kazakhstan.

```
SELECT student_id, name, address FROM Student WHERE address = 'Kazakhstan';
```

3) Find the name and major of students who has GPA greater or equal 3

```
SELECT Student.name, Student.gpa, Major.name FROM Student, Major
WHERE Student.student_id = Major.student_id AND Student.gpa >= 3;
```

4) Show name and Course name of students who choose the Database course;(Joins)

```
SELECT Student.name, Course.name FROM Course, Student INNER JOIN
Enroll ON Student.student_id = Enroll.student_id
WHERE (Course.course_id = Enroll.course_id AND Course.name = 'Database');
```

5) Find all information about students with highest GPA;(Aggregations)

```
SELECT * FROM Student WHERE Student.gpa = (SELECT MAX(gpa) FROM Student);
```

6) Find id and name of student who has age less than 18;

```
SELECT student_id, name, age FROM Student
WHERE age < 18;
```

7) Find how many Students has gpa lower than 1 (Aggregations)

```
SELECT COUNT(*) FROM Student WHERE Student.gpa = (SELECT COUNT(*) FROM
Student where gpa < 1)
```

8) Find id and name of professors who teaches FrontEnd course;(Joins)

```
SELECT Professor.name, Course.name
FROM Course, Professor RIGHT JOIN Section ON Professor.prof_id = Section.prof_id
WHERE(Section.course_id = Course.course_id)
OR Course.name = 'FrontEnd';
```

9) Find Professor's Experience, if it's greater than 10 years and less than 20 than print he is 'Experienced Professor' else if he's experience greater than 20 than print "Senior Professor"; else "Junior Professor"; (If else statements)

```
Select Professor.name, Professor.experience,
 CASE
        WHEN experience >= 10 AND experience < 20 THEN 'Experienced professor'
        WHEN experience >= 20 THEN 'Senior Professor'
        Else 'Junior Professor'
        END AS Experience
from Professor;
```

10) Create Function That gets bonus and mark of student as a parameter and returns the total(Functions)

```
DELIMITER $$
CREATE FUNCTION getTotal(bonus INT, mark INT)
RETURNS decimal(9,2)
DETERMINISTIC
BEGIN
    declare total DECIMAL(9,2);
    SET total = bonus + mark;
    return total;
END$$
DELIMITER ;
SELECT marks, bonus, getTotal(Grade.marks, Grade.bonus) FROM Grade;
```

11) Create Procedure that show fullname and experience of Professors in ascending order;(Procedure)

```
DELIMITER $$
CREATE PROCEDURE showProfessors()
BEGIN
  SELECT name, experience FROM Professor;
END$$
DELIMITER ;

CALL showProfessors();
```

12) Order Students, First by Their Age than by their GPA on Ascending order; (Ordering)

```
SELECT gpa, age FROM Student ORDER BY age, gpa ASC;
```

13) FindAssistant and Professor where assistant and professor works in one department(joins, as);

```
SELECT DISTINCT Professor.name as Professor, Assistant.name as Assistant,
Professor.dept_id, Assistant.dept_id
FROM Assistant INNER JOIN
Professor ON Assistant.dept_id = Professor.dept_id;
```

14) Update cours_name to Django where it’s Java;(update)

```
UPDATE Course SET Course.name = 'Django' WHERE Course.name = 'Java' LIMIT 1;
```

15) Create view that shows Students from Indonesia;

```
CREATE VIEW Indonesian_Students AS
SELECT *
FROM Student
WHERE address = 'Indonesia';
SELECT * FROM indonesian_students;
```

16) If there any student at university from Africa print YES, else print NO;(statements)

```
SELECT address,
   CASE
     WHEN EXISTS(SELECT address FROM Student
              WHERE address = 'Africa') THEN 'YES!'
       ELSE 'NO!'
   END
FROM Student WHERE address = 'Africa';
```

17) Find the name of professor who has experience more than others;(NOT IN)

```
SELECT Professor.name, MAX(experience)
FROM Professor
WHERE experience NOT IN (
   SELECT MAX(experience)
   FROM Professor
);
```

18) Union info from Assitants with info from Professor; (UNION, ALL)

```
SELECT *
    FROM Assistant
    INNER JOIN Professor
    ON Assistant.dept_id = Professor.dept_id
UNION ALL
    SELECT *
    FROM Professor
    INNER JOIN Assistant
    ON Professor.dept_id = Assistant.dept_id;
```

19) Find if Any assistant age is same with experience age of Professor; (ANY)

```
SELECT name, age
FROM Assistant
WHERE age = ANY
  (SELECT experience
  FROM Professor
  WHERE experience > 18);
```

20) Create View that shows Books with Genre Drama;(view)

```
CREATE VIEW drama_books AS
SELECT *
FROM Books
WHERE genre = ‘Drama’;
SELECT * FROM drama_books;
```

21) Find what kind of books does student's like more. Show genre and name of books;(join)

```
SELECT Books.name, Books.genre
FROM Books JOIN Borrow
ON Borrow.book_id = Books.book_id;
```

22) Create view that shows name and age of assistants(view);

```
CREATE VIEW info_assistants AS
SELECT Assistant.name, Assistant.age
FROM Assistant;
SELECT * FROM info_assistants;
```

23) Create Procedure that shows all books in library;

```
DELIMITER $$
   CREATE PROCEDURE showBooks()
   BEGIN
       SELECT Books.name FROM Books;
    END$$
DELIMITER ;
CALL showBooks();
```

24) Create Trigger that count each time when student enroll the course;(INSERT)

```
DELIMITER $$
CREATE TRIGGER insert_trigger
AFTER INSERT ON Enroll
FOR EACH ROW
        BEGIN
                UPDATE Counter SET insert_count = insert_count + 1;
        END;

SELECT * from Counter;
INSERT INTO Enroll (student_id, course_id) VALUES ('200535184', 'INF-202');
```

25) Create Trigger that counts each time when kick out student from university(DELETE)

```
DELIMITER $$
CREATE TRIGGER delete_trigger
AFTER DELETE ON Student
FOR EACH ROW
        BEGIN
                UPDATE Counter SET delete_count = delete_count + 1;
   END;

SELECT * from Counter;
DELETE FROM Student WHERE GPA = 1 LIMIT 1;
```

26) Create Trigger that saves name of student whose gpa become less than 1(UPDATE)

```
DELIMITER $$
CREATE TRIGGER update_trigger
AFTER UPDATE ON Student
FOR EACH ROW
        BEGIN
        IF(NEW.gpa <= 1) THEN
                INSERT INTO trigger_names(FullName) VALUES(NEW.name);
    END IF;
  END;


SELECT * FROM trigger_names;
UPDATE Student set Student.gpa = 1 where gpa = 2 LIMIT 1;
```