1. **Write SQL statements for the following queries in reference to relation Emp_time provided. [PU:2011 Fall,2015 Fall]**

Eid#	Name	Start_time	End_time
E101	Hari	10:15	18:00
E102	Malati	8:00	15:30
E103	Kalyan	9:30	17:00

i)create the table Eid# as primary key and insert the values provided

```
CREATE TABLE Emp_time (
  `Eid#` VARCHAR(10) PRIMARY KEY,
  Name VARCHAR(30),
  Start_time TIME,
  End_time TIME
);

INSERT INTO Emp_time  VALUES ('E101','Hari', '10:15:00', '18:00:00');
INSERT INTO Emp_time  VALUES  ('E102','Malati', '8:00:00', '15:30:00');
INSERT INTO Emp_time  VALUES    ('E103', 'Kalyan', '9:30:00', '17:00:00');
```

Note: *using backticks around the column name will allow us to use special characters. In above example we are using backtick in column name Eid#. Here special character # is used. i.e `Eid#`*

ii)**Display the name of the employee whose name start from letter 'M' and who work more than seven hours**

```
SELECT Name
FROM Emp_time
WHERE Name LIKE 'M%' AND TIMEDIFF(End_time, Start_time) >  '07:00:00';
```

iii)**Delete the entire content of the table so that new records can be inserted.**

```
TRUNCATE TABLE Emp_time;
```

2. Consider the relational database **[PU:2012 fall]**
   Employee(emp_name,street,city)
   Works(empname,cmpname,salary)
   Company(cmpname,city)
   Manages(empname,cmpname)

   Write SQL statement to:

i. Modify the database so that Amrit now lives in Naxal

```
UPDATE Employee
SET city = 'Naxal'
WHERE emp_name = 'Amrit';
```

ii. Delete all tuples in the works relation for employee of xyz corporation

```
DELETE FROM Works
WHERE cmpname = 'xyz corporation';
```

iii. Increase salary of all employees of ABC company by 10%

```
UPDATE Works
SET salary = salary * 1.1
WHERE cmpname = 'ABC';
```

iv. Display all company name located at city pokhara or kathmandu from company tables

```
SELECT cmpname
FROM Company
WHERE city = 'Pokhara' OR city= 'Kathmandu';
```

v. Display all empname who have salary greater than 5000 from works table

```
SELECT empname
FROM Works
WHERE salary > 5000;
```

3. Consider the following relation: **[PU:2012 Spring]**

Employee(empID,FirstName,LastName,address,DOB,sex,position,deptNo)
Department(deptNo,deptName,mgr,empID)
Project(projNo,projName,deptNo)
Work on(empID,projNo,hour worked)

Write SQL statements for the following

i) List the name and addresses of all employees who works for IT department

```
SELECT CONCAT(Employee.FirstName, ' ', Employee.LastName) AS name, Employee.address
FROM Employee, Department
WHERE Employee.deptNo = Department.deptNo
AND Department.deptName = 'IT';
```

ii) List the total hours worked by each employee, arranged in order of department number and within department alphabetically by employee surname

```
SELECT Employee.empID,  Employee.FirstName,Employee.LastName,Employee.deptNo, SUM(`hour worked`)
AS Totalhoursworked
FROM Employee, `Work on`
WHERE Employee.empID = `Work on`.empID
GROUP BY Employee.empID, Employee.FirstName, Employee.LastName, Employee.deptNo
ORDER BY Employee.deptNo, Employee.LastName;
```

iii) List the total number of employees in each department for those departments with more than 10 employees .

```
SELECT Department.deptName, COUNT(Employee.empID) AS TotalEmployees
FROM Employee, Department
WHERE Department.deptNo = Employee.deptNo
GROUP BY Department.deptName
HAVING COUNT(Employee.empID) > 10;
```

iv) List the project number ,project name and the number of employees who work in that project

```
 SELECT Project.projNo, Project.projName, COUNT(`Work on`.empID) AS num_employees
FROM Project ,`Work on`
WHERE Project.projNo = `Work on`.projNo
GROUP BY Project.projNo, Project.projName;
```

Note: *using backticks around the column name will allow us to use characters with blank space. Here `Work on` can be used.*

4. Consider the following relations: **[PU: 2014 fall]**

Employee(emp_name,street,city)
Works(emp_name,company,salary)
Company(comp_name,city)
Manages(emp_name,manager_name)
Write SQL statements for:

i. Find employee names that lives in the city same as the company city

```
SELECT Employee.emp_name
FROM Employee, Works, Company
WHERE Employee.emp_name = Works.emp_name
AND Works.company = Company.comp_name
AND Employee.city = Company.city;
```

ii. List all the employee details who earn more than 25000

```
SELECT  Employee.emp_name,Employee.street,Employee.city
FROM Employee, Works
WHERE Employee.emp_name = Works.emp_name
AND Works.salary > 25000;
```

iii. Update address of an employee ‘Sriyash’ to ’Pokhara’

```
UPDATE Employee
SET city = 'Pokhara'
WHERE emp_name = 'Sriyash';
```

iv. Create view for employee earns RS. 20,000 or more

```
CREATE VIEW HighEarningEmployee AS
SELECT  Employee.emp_name,Employee.street,Employee.city,Works.salary
FROM Employee, Works
WHERE Employee.emp_name = Works.emp_name
AND Works.salary >= 20000;
```

v. Delete all the employees from the table employee

```
DELETE FROM Employee;
```

5. Consider the relational database of figure given below, where primay keys are underlined. Given an expression in SQL for each of the following queries. **[PU:2014 spring]**

Employee(employee_name,street,city)
Works(employee_name,company_name,salary)
Company(company_name,ciy)
Manages(employee_name,manager_name)

i. modify the databases so that Ram now lives in kathmandu

```
UPDATE Employee
SET city = 'Kathmandu'
WHERE employee_name = 'Ram';
```

ii. Give all employees of First Bank Corporation a 10 percent raise

```
UPDATE Works
SET salary = salary * 1.1
WHERE company_name = 'First Bank Corporation';
```

iii. Give all managers of First Bank Corporation a 10 percent raise

```
UPDATE WORKS
SET salary = salary * 1.1
WHERE employee_name IN (SELECT manager_name  FROM Manages)
AND company_name= 'First Bank Corporation';
```

iv. Delete all in the work relation for employees of small bank corporation

```
DELETE FROM Works
WHERE company_name = 'Small Bank Corporation';
```

v. Find all employees who earn more than the average salary of all employees of their company

```
SELECT employee_name
FROM Works a
WHERE salary > (
   SELECT avg(salary)
   FROM Works b
   WHERE a.company_name = b.company_name
);
```

6. Suppose we are given the following table definitions with certain records in each table. **[PU: 2015 Spring]**

EMPLOYEE(EID,NAME,POST,AGE)
POST(POST_TITLE,SALARY)
PROJECT (PID,PNAME,DURATION,BUDGET)
WORK-IN (PID,EID,JOIN_DATE)

Write SQL statements for

i) **List the name of Employees whose age is greater than average age of all the employees**

```
SELECT NAME
FROM EMPLOYEE
WHERE AGE > (
  SELECT AVG(AGE)
  FROM EMPLOYEE
);
```

ii) Display all the employee numbers of those employees who are not working in any project

```
SELECT EID
FROM EMPLOYEE
WHERE EID NOT IN (
  SELECT EID
  FROM WORK_IN
);
```

iii**) List the name of employee and their salary who are working in the project 'DBMS'**

```
SELECT EMPLOYEE.NAME, POST.SALARY
FROM EMPLOYEE, POST,PROJECT,WORK_IN
WHERE PROJECT.PID=WORK_IN.PID AND
WORK_IN.EID=EMPLOYEE.EID  AND
EMPLOYEE.POST=POST.POST_TITLE  AND
PROJECT.PNAME='DBMS';
```

iv) update the database so that "Rishab now lives in "Butwal"

(This question cannot be solved from above relation. It seems question is incorrect)

7. Consider the relational schema: **[PU:2016 fall]**

Teacher (**TeacherID**,TeacherName,Offfice)

Write SQL statements for the following task:

i. To create a table from a table

```
CREATE TABLE Teacher
 (
  TeacherID INT PRIMARY KEY,
  TeacherName VARCHAR(50),
  Office VARCHAR(50)
);
```

ii. To eliminate duplicate rows

```
SELECT DISTINCT *
FROM Teacher;
```

iii. To add new column 'Gender' in the table

```
ALTER TABLE Teacher ADD Gender VARCHAR(10);
```

(Note: This query is for mariadb)

iv. To sort data in a table

```
SELECT *
FROM Teacher
ORDER BY TeacherName ASC;
```

v. To delete rows

```
DELETE FROM Teacher
WHERE TeacherID = 123;
```

In this example, rows with TeacherID equal to 123 are deleted. we can modify the condition in the WHERE clause to match your specific deletion criteria.

vi. Count the number of rows based in office

```
SELECT Office, COUNT(*) AS RowCount
FROM Teacher
GROUP BY Office;
```

8. Consider a simple relational database of Hospital management system.(Underlined attributes represent primary key attributes)

   Doctors (Doctor_ID,DoctorName,Department,Address,Salary)
   Patients(PatientID,Patient_Name,Address,Age,Gender)
   Hospitals (PatientID,DoctorID,HospitalName,Location)

   Write down the SQL statements for the following
   [PU:2016 spring][PU:2019 fall]

i. Display ID of patient admitted in hospital at pokhara and whose name ends with 'a'

```
SELECT PatientID
FROM Patients,Hospitals
WHERE Hospitals.PatientID = Patients.PatientID
AND HospitalName = 'Pokhara' AND Patient_Name LIKE '%a';
```

ii. Delete the records of Doctors whose salary is greater than average salary of doctors

```
DELETE FROM Doctors
WHERE Salary > (SELECT AVG(Salary) FROM Doctors);
```

iii. Increase salary of doctors by 18.5% who works in OPD department

```
UPDATE Doctors
SET Salary = Salary * 1.185
WHERE Department = 'OPD';
```

iv. Find the average salary of Doctors for each address who have average salary more than 55k.

```
SELECT Address, AVG(Salary) AS AverageSalary
FROM Doctors
GROUP BY Address
HAVING AVG(Salary) > 55000;
```

9. Write the SQL statements for the following Queries by reference to Liquors_info relation:

[PU:2017 fall]

Serial_No	Liquors	Start_year	Bottles	Ready_Year
1	Gorkha	1997	10	1998
2	Divine Wine	1998	5	2000
3	Old Durbar	1997	12	2001
4	Khukhuri Rum	1991	10	1992
5	Xing	1994	5	1995

i) **creates the Liquors_info relation**

```
CREATE TABLE Liquors_info
 (
   Serial_No INT,
   Liquors VARCHAR(50),
   Start_year YEAR,
   Bottles INT,
   Ready_Year YEAR
);
```

ii)insert the records in Liquor_info as above

```
INSERT INTO Liquors_info
VALUES (1, 'Gorkha', '1997', 10, '1998');

INSERT INTO Liquors_info
VALUES (2, 'Divine Wine', '1998', 5, '2000');

INSERT INTO Liquors_info
VALUES (3, 'Old Durbar', '1997', 12, '2001');

INSERT INTO Liquors_info
VALUES (4, 'Khukhuri Rum', '1991', 10, '1992');

INSERT INTO Liquors_info
VALUES (5, 'Xing', '1994', 5, '1995');
```

iii)**List all the records which were ready by 2000**

```
SELECT *
FROM Liquors_info
WHERE Ready_Year <= '2000';
```

iv)**Remove all records from database that required more than 2 years to get ready**

```
DELETE FROM Liquors_info

WHERE (Ready_Year -Start_year) > 2;
```

10. Consider the following three relations **[PU:2018 fall]**

Doctor(Name,age,address)
Works(Name,Depart_no,salary)
Department(Depart_no,dept_name,floor,room)

Write down the SQL statement for the following

i)**Display the name of doctor who do not work in any department**

```
SELECT Name
FROM Doctor
WHERE Name NOT IN (
  SELECT Name
  FROM Works
);
```

ii) **Modify the database so that Dr.Hari Lives in pokhara**

```
UPDATE Doctor
SET address = 'Pokhara'
WHERE Name = 'Dr. Hari';
```

iii) **Delete all records of Doctor working OPD department**

```
DELETE FROM Doctor
WHERE Name IN
( SELECT Works.Name
FROM Works ,Department
WHERE Works.Depart_no = Department.Depart_no
 AND Department.dept_name = 'OPD'
);
```

iv) **Display the name of doctors who works in at least two department**

```
SELECT Doctors.Name
FROM Doctor,Works
WHERE Doctors. Name = Works.Name
GROUP BY Doctors.Name
HAVING COUNT(DISTINCT Works.Depart_no) >= 2;
```

11. Write SQL statements for the following: [PU:2018 spring]

i) create a table named Vehicle with veh_number as primary key and following attributes:

veh_type,veh_brand,veh_year,veh_mileage,veh_owner,veh_photo,veh_price

```
CREATE TABLE Vehicle (
  veh_number INT PRIMARY KEY,
  veh_type VARCHAR(50),
  veh_brand VARCHAR(50),
  veh_year YEAR,
  veh_mileage int,
  veh_owner VARCHAR(50),
  veh_photo LONGBLOB,
  veh_price DECIMAL(12, 2)
);
```

ii)Enter a full detailed information of a vehicle

```
INSERT INTO Vehicle
VALUES (12376, 'Sedan', 'Toyota', 2022, 55, 'Hari Pokhrel', LOAD_FILE('C:\\Users\\Downloads\\veh1.jpg'),
18500.75);
```

iii)Increment a vehicle price by 10,000

```
UPDATE Vehicle
SET veh_price = veh_price + 10000
WHERE veh_number = 12376;
```

This SQL UPDATE statement will find the vehicle with veh_number 12376 in the "Vehicle" table and increase its veh_price by 10,000.

iv)**Remove all vehicle's records whose brand contains character 'o' second position**

```
DELETE FROM Vehicle
WHERE veh_brand LIKE '_o%';
```

v)Display the total price of all vehicles

```
SELECT SUM(veh_price) AS total_price
FROM Vehicle;
```

vi)create a view a from above table

```
CREATE VIEW Low_price_vehicle AS
SELECT veh_number, veh_type, veh_brand, veh_year, veh_price
FROM Vehicle
WHERE veh_price < 150000;
```

vii)Display details of vehicles ordering on descending manner in brand and by mileage when brand matches

```
SELECT *
FROM Vehicle
ORDER BY veh_brand DESC, veh_mileage DESC;
```

viii) change data type of year to datetime

```
ALTER TABLE Vehicle
MODIFY COLUMN veh_year DATETIME;
```

12. Write a SQL statements for the following **[PU:2020 fall]**

i)Create a table named Automotor with chasis_number as primary key and following attributes.

veh_brand,veh_name,veh_model,veh_year,veh_cost,veh_color,veh_weight

```
CREATE TABLE Automotor (
  chasis_number INT PRIMARY KEY,
  veh_brand VARCHAR(50),
  veh_name VARCHAR(50),
  veh_model VARCHAR(50),
  veh_year YEAR,
  veh_cost DECIMAL(10,2),
  veh_color VARCHAR(10),
  veh_weight DECIMAL(10,2)
);
```

ii)**Enter a full detailed information of automotor**

```
INSERT INTO Automotor

VALUES (1, 'Toyota', 'Corolla', 'XE', 2020, 25000.00, 'Red', 1200.50);
```

iii)Change any Automotor's year to 2019

```
UPDATE Automotor
SET veh_year = 2019
WHERE chasis_number=125;
```

iv)Remove all Automotor's records whose model contains 'i' in last position

```
DELETE FROM Automotor

WHERE veh_model LIKE '%i';
```

v)Display the total cost of all vehicles of the table Automotor

```
SELECT SUM(veh_cost) AS total_cost
FROM Automotor;
```

vi) Create a view from above table having vehicle only red color

```
CREATE VIEW RedVehicles AS
SELECT * FROM Automotor
 WHERE veh_color = 'red';
```

vii) Display details of Automotor ordering on descending manner by brand name and ascending order on model when brand matches

```
SELECT *
FROM Automotor
ORDER BY veh_brand DESC, veh_model ASC;
```

viii) change data type of veh_color so that it only takes one character

```
ALTER TABLE Automotor
MODIFY COLUMN veh_color CHAR(1);
```

(Note: This query is for mariaDB. Syntax may vary in different DBMS)

13. Write SQL statement for the following schemas (underline indicates primary key) **[PU:2020 spring]**

Employee(**Emp_No**,Name,Address)
Project(**PNo**,Pname)
Workon(**Emp_No**,PNo)
Part(**Partno**,Part_name,Qty_on_hand)
Use(**Emp_No**,**PNo**,**Partno**,Number)

a. Listing all the employee details who are not working yet

```
SELECT Emp_No, Name, Address
FROM Employee
WHERE Emp_No NOT IN (SELECT DISTINCT Empno FROM Workon);
```

b. Listing Part_name and Qty_on_hand those were used in DBMS project

```
SELECT Part.Part_name, Part.Qty_on_hand
FROM Part,Use,Project
WHERE Part.Partno = Use.Partno
AND Use.PNo=Project.PNo
AND Project.pname='DBMS';
```

c. List the name of the projects that are used by employee from London

```
SELECT DISTINCT Project.Pname
FROM Project,Use,Employee
WHERE Project.PNo = Use.PNo AND
Use.Emp_No = Employee.Emp_No AND
Employee.Address = 'London';
```

d. Modify the database so that Jones now lives in 'USA'

```
UPDATE Employee
SET Address = 'USA'
WHERE Name = 'Jones';
```

e. Update address of an employee 'Japan' to 'USA'

```
UPDATE Employee
SET Address = 'USA'
WHERE Address = 'Japan';
```

14. Let us consider the following relation

Sailors (sid,sname,rating,age)
Boats(bid, bname,color)
Reserves(sid,bid,day)

Write a SQL statements for the following

i)Find the records of sailors who have reserved boat number 103(bid=103)

```
SELECT Sailors.sid,Sailors.sname,Sailors.rating,Sailors.age
FROM Sailors , Reserves
WHERE Sailors.sid = Reserves.sid
AND Reserves.bid = 103;
```

ii)Update the color of the boat ,where bid is 104,into green

```
UPDATE Boats
SET color = 'green'
WHERE bid = 104;
```

iii) find the name of sailors who have reserved a red or green boat

```
SELECT  Sailors.sname
FROM Sailors, Reserves, Boats
WHERE Sailors.sid = Reserves.sid
AND Reserves.bid = Boats.bid
AND Boats.color IN ('red', 'green');
```

iv)find the name of sailors who have reserved boat number 103 on day 5

```
SELECT Sailors.sname
FROM Sailors , Reserves
WHERE Sailors.sid = Reserves.sid
 AND Reserves.bid = 103
 AND Reserves.day = 5;
```

v)find the name of sailors whose name is not 'Ram'

```
SELECT  sname
FROM Sailors
WHERE sname != 'Ram';
```

vi) find the name of all boats

```
SELECT bname

FROM Boats;
```

15) **Write the SQL statements for the following queries by reference of Hotel_details relation**

[PU:2019 spring]

Hotel_id	Hotel_name	Estb_year	Hotel_star	Hotel_worth
1	Hyatt	2047	Five	15M
2	Hotel ktm	2043	Three	5M
3	Fullbari	2058	Five	20M
4	Yak and Yeti	2052	Four	11M
5	Hotel chitwan	2055	Three	7M

i) **create a database named hotel and table relation**

```
-- Create the database

CREATE DATABASE hotel;

-- Switch to the hotel database

USE hotel;
```

-- Create the table Hotel_details

```
CREATE TABLE Hotel_details (
  Hotel_id INT,
  Hotel_name VARCHAR(50),
  Estb_year INT,
  Hotel_star VARCHAR(10),
  Hotel_worth BIGINT
);
```

ii)create a view named price which shows hotel name and its worth

```
CREATE VIEW price AS
SELECT Hotel_name, Hotel_worth
FROM Hotel_details;
```

iii) modify the data so that hotel chitwan is now four star level

```
UPDATE Hotel_details
SET Hotel_star = 'Four'
WHERE Hotel_name = 'Hotel_chitwan';
```

iv)delete the records of all hotels having worth more than 9M

```
DELETE FROM Hotel_details
WHERE Hotel_worth > 9000000;
```

16. Consider the following relation

Orders(order_id,product_name,price,quantity,order_date,delivery_date)

1) Create table orders

```
CREATE TABLE Orders (
   order_id INT PRIMARY KEY,
   product_name VARCHAR(50) ,
   price DECIMAL(10, 2) ,
   quantity INT ,
   order_date DATE ,
   delivery_date DATE
);
```

2) Now insert any 8 records

```
INSERT INTO Orders
VALUES (1, 'T-shirt', 25.99, 2, '2023-07-15', '2023-07-25');

INSERT INTO Orders
VALUES (2, 'Jeans', 49.95, 1, '2023-07-17', '2023-07-20');

INSERT INTO Orders
VALUES (3, 'Shoes', 69.50, 1, '2023-07-20', '2023-07-30');

INSERT INTO Orders
VALUES (4, 'Sunglasses', 12.75, 3, '2023-07-22', '2023-07-28');

INSERT INTO Orders
VALUES (5, 'Backpack', 34.99, 2, '2023-07-25', '2023-07-29');

INSERT INTO Orders
VALUES (6, 'Headphones', 59.99, 1, '2023-07-29', '2023-08-05');

INSERT INTO Orders
VALUES (7, 'Smartphone', 299.99, 2, '2023-07-29', '2023-11-01');

INSERT INTO Orders
VALUES (8, 'Laptop', 799.95, 1, '2023-07-29', '2025-08-01');
```

3) Retrieve all orders placed on a 2023-07-15

```
SELECT *
 FROM Orders
WHERE order_date = '2023-07-15';
```

4) Find the number of days that required to delivered shoes

```
SELECT DATEDIFF(delivery_date, order_date) AS delivery_time
FROM Orders
where product_name='shoes';
```

5) Find all the orders that is received from '2023-07-15' to '2023-07-25'

```
SELECT *
FROM Orders
WHERE order_date BETWEEN '2023-07-15' AND '2023-07-25';
```

6) find all the orders that is received today

```
SELECT *
FROM Orders
WHERE order_date = CURDATE();
```

7) Calculate the average number of days it takes to deliver a orders

```
SELECT AVG(DATEDIFF(delivery_date, order_date)) AS avg_delivery_time
FROM Orders;
```

Here, in DATDIFF() function we have passed two parameters that is

DATEDIFF(date1, date2)

This will returns date difference in terms of number of days (date2-date1)

This is for if we run query on **MySQL DBMS.**

But sometimes it is necessary to find out date difference in terms of number of month, week, year, quarter etc. In such case three parameters need to passed three parameters.

Syntax

```
DATEDIFF(interval, date1, date2)
```

Parameter Values

Parameter	Description
interval	Required. The part to return. Can be one of the following values: • year, yyyy, yy = Year • quarter, qq, q = Quarter • month, mm, m = month • dayofyear = Day of the year • day, dy, y = Day • week, ww, wk = Week • weekday, dw, w = Weekday • hour, hh = hour • minute, mi, n = Minute • second, ss, s = Second • millisecond, ms = Millisecond
date1, date2	Required. The two dates to calculate the difference between

Note:

- DATEDIFF() function with two parameters are supported in MYSQL DBMS.
- DATEDIFF() function with three parameters are supported in MS SQL Server DBMS

If you want test query in different DBMS ,you can follow this link

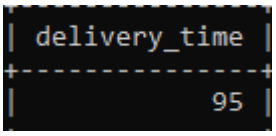
http://sqlfiddle.com/#!18

8) Find the number of months required to deliver smartphone

```
SELECT DATEDIFF(delivery_date, order_date) AS delivery_time
FROM Orders
where product_name='smartphone';
```

If we run this query in MySQL DBMS.

DATEDIFF() function will returns 95 for this query by considering above relations.



delivery_time
95

DATEDIFF() with three parameters are not supported in MySQL DBMS.

This query can be re-written as follows by passing three parameters in MS SQL server DBMS.

```
SELECT DATEDIFF(month, order_date,delivery_date) AS delivery_time
FROM Orders
where product_name='smartphone';
```

Note: you can write DATEDIFF() function with three parameters in exam.

9) Find the number of weeks required to deliver smartphone

```
SELECT DATEDIFF(week, order_date,delivery_date) AS delivery_time
FROM Orders
where product_name='smartphone';
```

10) Find the products that required more than 2 month to delivered

```
SELECT product_name
FROM orders
WHERE DATEDIFF(month,order_date,delivery_date)>2;
```

11) Find the products that required more than 3 weeks to delivered

```
SELECT product_name
FROM orders
WHERE DATEDIFF(week,order_date,delivery_date)>3;
```

12. **Find the products that required more than 1 years to delivered.**

```
SELECT product_name
FROM orders
WHERE DATEDIFF(year,order_date,delivery_date)>1;
```

The SQL SELECT TOP Clause

- ✓ The SELECT TOP clause is used to specify the number of records to return.
- ✓ The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

Note: Not all database systems support the SELECT TOP clause. MySQL supports the LIMIT clause to select a limited number of records, while Oracle uses FETCH FIRST n ROWS ONLY and ROWNUM.

My SQL syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
LIMIT number;
```

Find the top 3 records from orders from orders

```
SELECT *
FROM orders
LIMIT 3;
```

Create relational database for the Department of computer Engineering (DOCE) of pokhara university. Your database should have at least three relations Describe referential integrity constraint based on above database of DOCE.[PU:2017 spring]

Based on the Department of Computer Engineering (DOCE) of Pokhara University, we can create a relational database with following relations: "Student," "Faculty ,"Course" "Enroll"

Student(student_id,student_name,email,address)
Faculty(faculty_id,faculty_name,qualification)
Course(course_id ,course_name,course_description,faculty_id)
Enroll(enroll_id,student_id,course_id,enrollment_date)

Now creating tables

```
CREATE TABLE Student (
   student_id INT PRIMARY KEY,
   student_name VARCHAR(50),
   email VARCHAR(50),
   address VARCHAR(100)
);
CREATE TABLE Faculty (
   faculty_id INT PRIMARY KEY,
   faculty_name VARCHAR(50),
   qualification VARCHAR(50)
);
CREATE TABLE Course (
   course_id INT PRIMARY KEY,
   course_name VARCHAR(50),
   course_description LONGTEXT,
   faculty_id INT,
   FOREIGN KEY (faculty_id) REFERENCES Faculty (faculty_id)
);
CREATE TABLE Enroll (
   enroll_id INT PRIMARY KEY,
   student_id INT,
   course_id INT,
   enrollment_date DATE,
   FOREIGN KEY (student_id) REFERENCES Student(student_id),
   FOREIGN KEY (course_id) REFERENCES Course(course_id)
);
```

Based on the provided relations, the following are the foreign key integrity constraints that can be applied to maintain referential integrity:

Course table:
The faculty_id column in the Course table is a foreign key referencing the faculty_id column in the Faculty table. This ensures that the faculty_id value in the Course table must exist in the Faculty table.

Enroll table:

The student_id column in the Enroll table is a foreign key referencing the student_id column in the Student table. This ensures that the student_id value in the Enroll table must exist in the Student table.
The course_id column in the Enroll table is a foreign key referencing the course_id column in the Course table. This ensures that the course_id value in the Enroll table must exist in the Course table.

These foreign key constraints help maintain data integrity by enforcing the relationships between the tables. They prevent the insertion of invalid values that do not exist in the referenced tables, ensuring the consistency of the data across the relations.

Note: You can draw schema diagram for above relations as well.