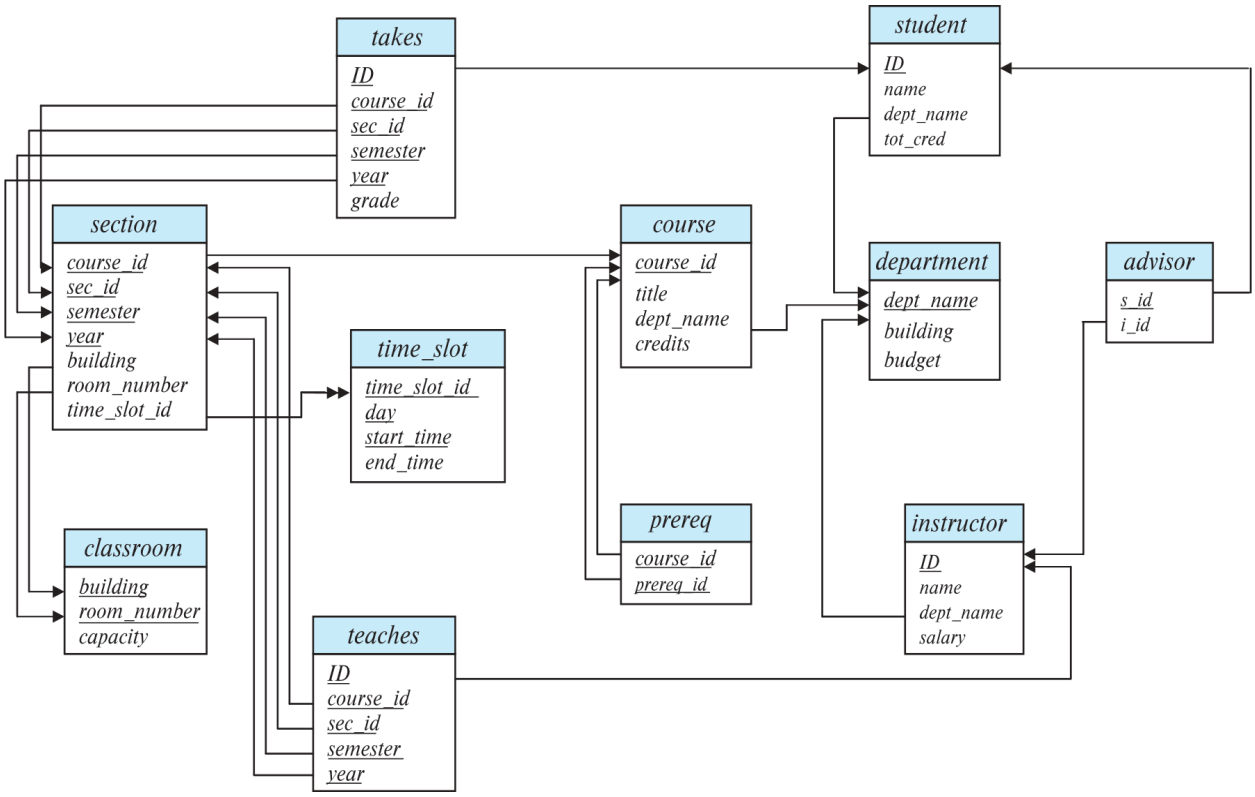
# ***Assignment-3***

1. ***Create a database named University.***
2. ***Create all the tables of given database schema following all the constraints(PK, FK) given in the schema.***




3. ***Insert minimum 10 rows in each table.***
4. ***RUN all the queries covered in the slides of lecture-2 and lecture-3 shared with you.***

5. ***Based on the tables Sales and Products, write SQL queries for the following questions related to the concept of Views in DBMS.***

### *Q5.1. Simple View*

*Create a view named Product_Sales_View that shows the product_name, category, quantity_sold, and total_price for each sale by joining the Sales and Products tables.*

- *Then, query this view to list all sales of products in the 'Electronics' category.*

***Q5.2. Aggregated View***

*Define a view named Category_Sales_Summary that shows the category and the total revenue generated for each product category.*

- *Write a query on this view to find which category generated the highest revenue.*

***Q5.3. Nested View***

*First, create a view named Daily_Sales_Total that shows sale_date and the sum of total_price for each date.*
*Then, create another view named High_Sales_Days that selects only those days where total sales exceeded ₹50,000.*

***Q5.4. Security View (Column Restriction)***

*Create a view named Product_List_View that shows only product_id, product_name, and category from the Productstable (hiding the unit_price).*

- *Why might such a view be useful in a real-world scenario?*

***Q5.5. Materialized View Simulation***

*Since MySQL does not support materialized views directly, simulate one by creating a table Monthly_Category_Salesthat stores the monthly total sales per category (from Sales and Products).*

- *Explain how you would refresh this table when new sales are inserted.*

***Q6. Joins***

***Q6.1*** *Write a query to list the names of students along with the course IDs of the courses they have taken using a* ***natural join*** *between Student and Takes.*

***Q6.2*** *Write a query to find all instructors and the titles of the courses they teach using an* ***inner join*** *between Instructor and Course.*

***Q6.3*** *Write a query to display all courses along with their prerequisites using a* ***left outer join*** *between Course and Prereq.*

***Q8. Integrity Constraints***

***Q8.1*** *Alter the Student table to add a* ***check constraint*** *ensuring that total credits (tot_cred) cannot be negative.*

***Q8.2*** *Add a* ***foreign key constraint*** *on the Course table ensuring that every dept_name must exist in the Department table with* ***ON DELETE CASCADE****.*

***Q9. Indexes***

***Q9.1*** *Create an index on the Instructor table for the dept_name column. Why might this index be useful?*

***Q9.2*** *Create an index on the Takes(ID) attribute and explain how it would improve query performance when retrieving courses for a particular student.*

***Q10. Authorization***

***Q10.1*** *Write SQL statements to grant* ***SELECT*** *privilege on the Instructor table to user Amit.*

***Q10.2*** *Create a role named Teaching_Assistant and grant it the privilege to* ***SELECT*** *from Takes and* ***UPDATE*** *Student total credits.*

***Q10.3*** *Write a query to revoke the Teaching_Assistant role from a user.*