**University Database**

A database management system (DBMS) is a collection of interconnected data and a set of applications for accessing that data. A database is a collection of data that contains information relevant to a company. A database management system's primary purpose is to provide an easy and efficient method for storing and retrieving database information.

Full Schema

classroom (building, room\_number, capacity)

department(dept\_name, building, budget)

course(course\_id, title, dept\_name, credits)

instructor(ID, name, dept\_name, salary)

section(course\_id, sec\_id, semester, year, building, room\_number, time\_slot\_id)

teaches(ID, course\_id, sec id, semester, year)

student(ID, name, dept name, tot\_cred)

takes(ID, course\_id, sec\_id, semester, year, grade)

advisor(s\_ID, i\_ID)

time slot(time\_slot\_id, day, start\_time, end\_time)

prereq(course\_id, prereq\_id)

Figure: Schema of the university database.

The university database is a critical system designed to handle interconnected information necessary for both academic and administrative functions. It encompasses key components such as **Classroom**, which holds data about physical spaces, including building names and room capacities; **Department**, which manages academic units based on names, locations, and budgets; and **Course**, which provides details like course IDs, titles, and credit hours.

The **Instructor** entity captures faculty details and associates them with their respective departments, while the **Section** entity specifies course offerings for each semester, along with classroom allocations. The **Teaches** table defines the connection between instructors and the sections they lead, ensuring clarity in teaching assignments.

Students are represented in the **Student** entity, which stores information such as student IDs, names, and total earned credits. The **Takes** table monitors student course enrollments and records their grades. Moreover, the **Advisor** entity oversees student-faculty advising relationships, improving academic guidance and support.

The **Time Slot** entity schedules classes, while the **Prereq** table outlines the prerequisites for courses. This comprehensive database streamlines the management of academic processes, maintains data consistency, and supports the institution’s educational objectives, thereby enriching the experiences of both students and faculty.

**Important Considerations**

1. **Data Integrity**: Ensure foreign key relationships are correctly implemented to maintain consistency across tables, such as linking dept\_name in course to the department table.

2. **Normalization**: Regularly review the schema to minimize redundancy and avoid duplicating information across multiple tables

3. **Access Control**: Implement strict access controls to protect sensitive data, especially student records and financial information.

4. **Backup and Recovery**: Establish a regular backup schedule and a clear recovery process to safeguard data against loss.

5. **Performance Optimization**: Monitor and optimize database performance to efficiently handle high query volumes, particularly during registration periods.

6. **Scalability**: Design the schema to accommodate growth, allowing for easy addition of courses, students, and departments.

7. **Documentation**: Maintain clear documentation of the schema and entity relationships to facilitate maintenance and future development.

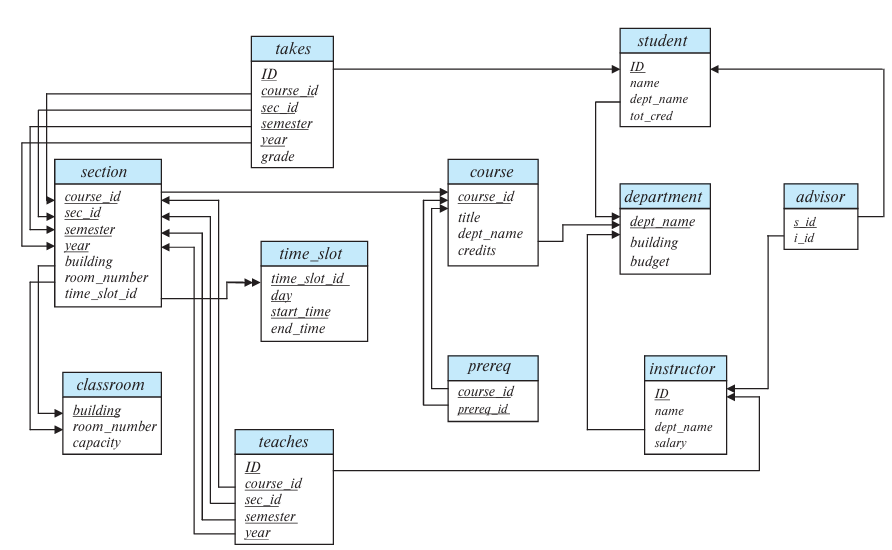
**Schema Diagram for the University Database**

Figure: Schema diagram for the university database.

**E-R Diagram for University Database**

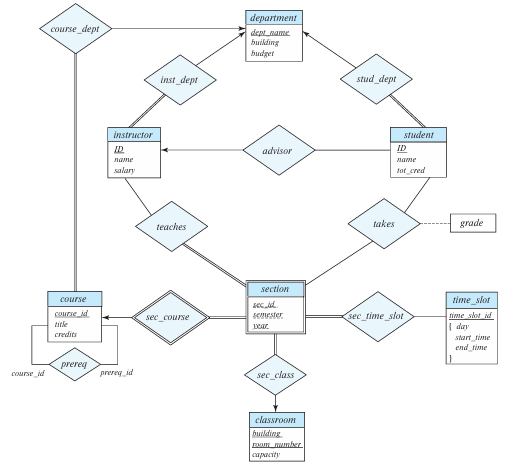
****

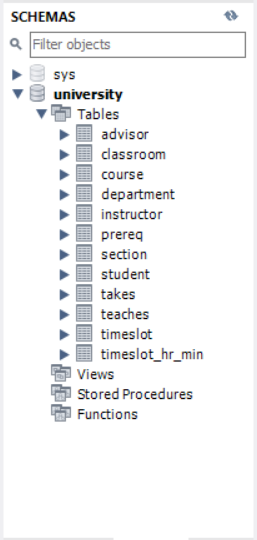
Figure: E-R diagram for a university database.

**Problem no –** 01

**Problem Name –** Write SQL queries using integrity constraints to create tables for a database.

**Query –**

CREATE TABLE classroom (  
 building VARCHAR(15),  
 room\_number VARCHAR(7),  
 capacity NUMERIC(4,0),  
 PRIMARY KEY (building, room\_number));  
CREATE TABLE department (  
 dept\_name VARCHAR(20),  
 building VARCHAR(15),  
 budget NUMERIC(12,2) CHECK (budget > 0),  
 PRIMARY KEY (dept\_name),  
 FOREIGN KEY (building) REFERENCES classroom(building));  
CREATE TABLE course (  
 course\_id VARCHAR(7),  
 title VARCHAR(50),  
 dept\_name VARCHAR(20),  
 credits NUMERIC(2,0) CHECK (credits > 0),  
 PRIMARY KEY (course\_id),  
 FOREIGN KEY (dept\_name) REFERENCES department(dept\_name) ON DELETE SET NULL );  
CREATE TABLE instructor (  
 ID VARCHAR(5),  
 name VARCHAR(20) NOT NULL,  
 dept\_name VARCHAR(20),  
 salary NUMERIC(8,2) CHECK (salary > 29000),  
 PRIMARY KEY (ID),  
 FOREIGN KEY (dept\_name) REFERENCES department(dept\_name) ON DELETE SET NULL );  
CREATE TABLE section (  
 course\_id VARCHAR(8),  
 sec\_id VARCHAR(8),  
 semester VARCHAR(6) CHECK (semester IN ('Fall', 'Winter', 'Spring', 'Summer')),  
 year NUMERIC(4,0) CHECK (year > 1701 AND year < 2100),  
 building VARCHAR(15),  
 room\_number VARCHAR(7),  
 time\_slot\_id VARCHAR(4),  
 PRIMARY KEY (course\_id, sec\_id, semester, year),  
 FOREIGN KEY (course\_id) REFERENCES course(course\_id) ON DELETE CASCADE,  
 FOREIGN KEY (building, room\_number) REFERENCES classroom(building, room\_number) ON DELETE SET NULL );  
CREATE TABLE teaches (  
 ID VARCHAR(5),  
 course\_id VARCHAR(8),  
 sec\_id VARCHAR(8),  
 semester VARCHAR(6),  
 year NUMERIC(4,0),  
 PRIMARY KEY (ID, course\_id, sec\_id, semester, year),  
 FOREIGN KEY (course\_id, sec\_id, semester, year) REFERENCES section(course\_id, sec\_id, semester, year) ON DELETE CASCADE,  
 FOREIGN KEY (ID) REFERENCES instructor(ID) ON DELETE CASCADE );  
CREATE TABLE student (  
 ID VARCHAR(5),  
 name VARCHAR(20) NOT NULL,  
 dept\_name VARCHAR(20),  
 tot\_cred NUMERIC(3,0) CHECK (tot\_cred >= 0),  
 PRIMARY KEY (ID),  
 FOREIGN KEY (dept\_name) REFERENCES department(dept\_name) ON DELETE SET NULL );  
CREATE TABLE takes (  
 ID VARCHAR(5),  
 course\_id VARCHAR(8),  
 sec\_id VARCHAR(8),  
 semester VARCHAR(6),  
 year NUMERIC(4,0),  
 grade VARCHAR(2),  
 PRIMARY KEY (ID, course\_id, sec\_id, semester, year),  
 FOREIGN KEY (course\_id, sec\_id, semester, year) REFERENCES section(course\_id, sec\_id, semester, year) ON DELETE CASCADE,  
 FOREIGN KEY (ID) REFERENCES student(ID) ON DELETE CASCADE );  
CREATE TABLE advisor (  
 s\_ID VARCHAR(5),  
 i\_ID VARCHAR(5),  
 PRIMARY KEY (s\_ID, i\_ID),  
 FOREIGN KEY (i\_ID) REFERENCES instructor(ID),  
 FOREIGN KEY (s\_ID) REFERENCES student(ID) ON DELETE CASCADE );  
CREATE TABLE prereq (  
 course\_id VARCHAR(8),  
 prereq\_id VARCHAR(8),  
 PRIMARY KEY (course\_id, prereq\_id),  
 FOREIGN KEY (course\_id) REFERENCES course(course\_id) ON DELETE CASCADE,  
 FOREIGN KEY (prereq\_id) REFERENCES course(course\_id) );  
CREATE TABLE timeslot (  
 time\_slot\_id VARCHAR(4),  
 day VARCHAR(1) CHECK (day IN ('M', 'T', 'W', 'R', 'F', 'S', 'U')),  
 start\_time TIME,  
 end\_time TIME,  
 PRIMARY KEY (time\_slot\_id, day, start\_time) );  
CREATE TABLE timeslot\_hr\_min (  
 time\_slot\_id VARCHAR(4),  
 day VARCHAR(1),  
 start\_hr NUMERIC(2) CHECK (start\_hr >= 0 AND start\_hr < 24),  
 start\_min NUMERIC(2) CHECK (start\_min >= 0 AND start\_min < 60),  
 end\_hr NUMERIC(2) CHECK (end\_hr >= 0 AND end\_hr < 24),  
 end\_min NUMERIC(2) CHECK (end\_min >= 0 AND end\_min < 60),  
 PRIMARY KEY (time\_slot\_id, day, start\_hr, start\_min)  
);

**Output –**

**Conclusion –** In conclusion we can say that we have successfully created the tables of the pre-designed university database.

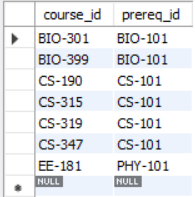
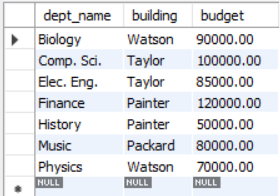
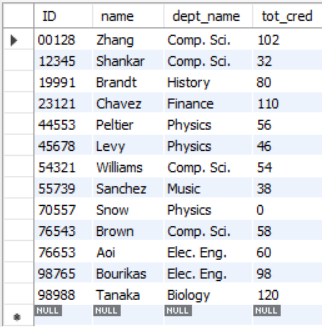
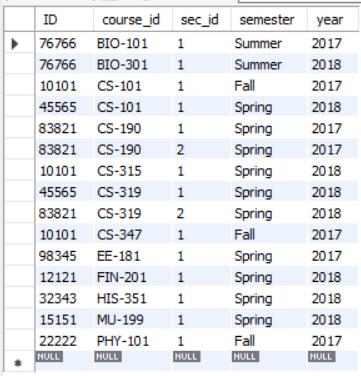
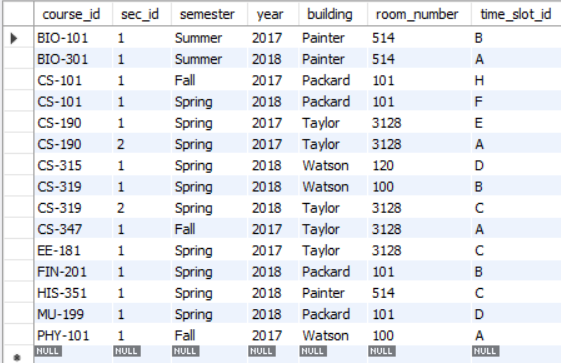
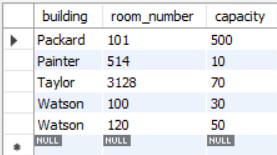
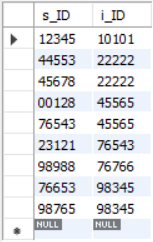
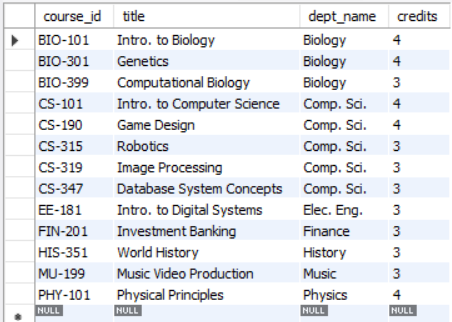
**Problem no –** 02

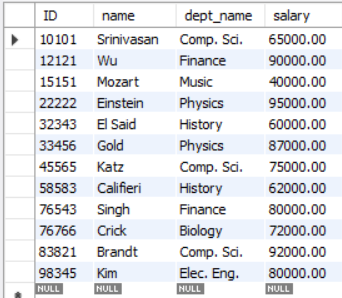
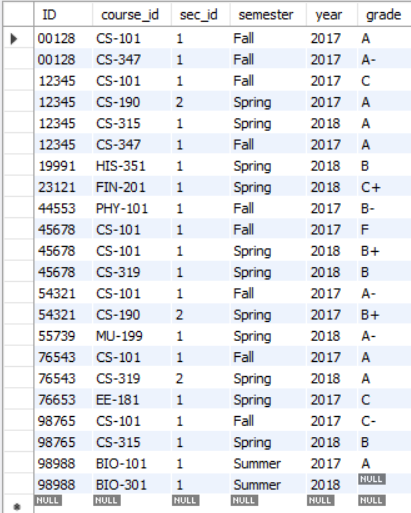
**Problem Name –** Write SQL queries to insert values into tables in the university database.

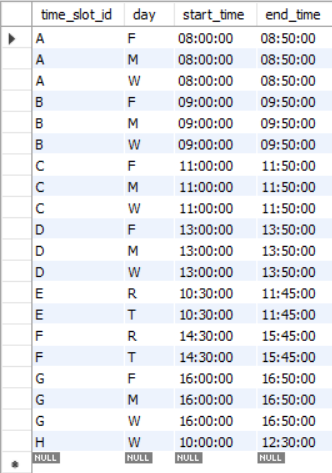
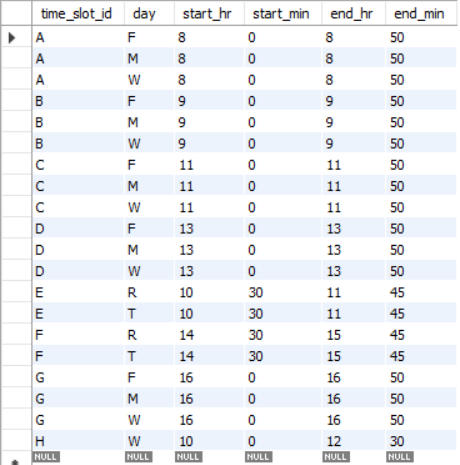
**Query –**

INSERT INTO classroom (building, room\_number, capacity) VALUES  
('Packard', '101', 500),  
('Painter', '514', 10),  
('Taylor', '3128', 70),  
('Watson', '100', 30),  
('Watson', '120', 50);  
INSERT INTO department (dept\_name, building, budget) VALUES  
('Biology', 'Watson', 90000),  
('Comp. Sci.', 'Taylor', 100000),  
('Elec. Eng.', 'Taylor', 85000),  
('Finance', 'Painter', 120000),  
('History', 'Painter', 50000),  
('Music', 'Packard', 80000),  
('Physics', 'Watson', 70000);   
INSERT INTO course (course\_id, title, dept\_name, credits) VALUES  
('BIO-101', 'Intro. to Biology', 'Biology', 4),  
('BIO-301', 'Genetics', 'Biology', 4),  
('BIO-399', 'Computational Biology', 'Biology', 3),  
('CS-101', 'Intro. to Computer Science', 'Comp. Sci.', 4),  
('CS-190', 'Game Design', 'Comp. Sci.', 4),  
('CS-315', 'Robotics', 'Comp. Sci.', 3),  
('CS-319', 'Image Processing', 'Comp. Sci.', 3),  
('CS-347', 'Database System Concepts', 'Comp. Sci.', 3),  
('EE-181', 'Intro. to Digital Systems', 'Elec. Eng.', 3),  
('FIN-201', 'Investment Banking', 'Finance', 3),  
('HIS-351', 'World History', 'History', 3),  
('MU-199', 'Music Video Production', 'Music', 3),  
('PHY-101', 'Physical Principles', 'Physics', 4);  
INSERT INTO instructor (ID, name, dept\_name, salary) VALUES  
('10101', 'Srinivasan', 'Comp. Sci.', 65000),  
('12121', 'Wu', 'Finance', 90000),  
('15151', 'Mozart', 'Music', 40000),  
('22222', 'Einstein', 'Physics', 95000),  
('32343', 'El Said', 'History', 60000),  
('33456', 'Gold', 'Physics', 87000),  
('45565', 'Katz', 'Comp. Sci.', 75000),  
('58583', 'Califieri', 'History', 62000),  
('76543', 'Singh', 'Finance', 80000),  
('76766', 'Crick', 'Biology', 72000),  
('83821', 'Brandt', 'Comp. Sci.', 92000),  
('98345', 'Kim', 'Elec. Eng.', 80000);  
INSERT INTO section (course\_id, sec\_id, semester, year, building, room\_number, time\_slot\_id) VALUES  
('BIO-101', '1', 'Summer', 2017, 'Painter', '514', 'B'),  
('BIO-301', '1', 'Summer', 2018, 'Painter', '514', 'A'),  
('CS-101', '1', 'Fall', 2017, 'Packard', '101', 'H'),  
('CS-101', '1', 'Spring', 2018, 'Packard', '101', 'F'),  
('CS-190', '1', 'Spring', 2017, 'Taylor', '3128', 'E'),  
('CS-190', '2', 'Spring', 2017, 'Taylor', '3128', 'A'),  
('CS-315', '1', 'Spring', 2018, 'Watson', '120', 'D'),  
('CS-319', '1', 'Spring', 2018, 'Watson', '100', 'B'),  
('CS-319', '2', 'Spring', 2018, 'Taylor', '3128', 'C'),  
('CS-347', '1', 'Fall', 2017, 'Taylor', '3128', 'A'),  
('EE-181', '1', 'Spring', 2017, 'Taylor', '3128', 'C'),  
('FIN-201', '1', 'Spring', 2018, 'Packard', '101', 'B'),  
('HIS-351', '1', 'Spring', 2018, 'Painter', '514', 'C'),  
('MU-199', '1', 'Spring', 2018, 'Packard', '101', 'D'),  
('PHY-101', '1', 'Fall', 2017, 'Watson', '100', 'A');  
INSERT INTO teaches (ID, course\_id, sec\_id, semester, year) VALUES  
('10101', 'CS-101', '1', 'Fall', 2017),  
('10101', 'CS-315', '1', 'Spring', 2018),  
('10101', 'CS-347', '1', 'Fall', 2017),  
('12121', 'FIN-201', '1', 'Spring', 2018),  
('15151', 'MU-199', '1', 'Spring', 2018),  
('22222', 'PHY-101', '1', 'Fall', 2017),  
('32343', 'HIS-351', '1', 'Spring', 2018),  
('45565', 'CS-101', '1', 'Spring', 2018),  
('45565', 'CS-319', '1', 'Spring', 2018),  
('76766', 'BIO-101', '1', 'Summer', 2017),  
('76766', 'BIO-301', '1', 'Summer', 2018),  
('83821', 'CS-190', '1', 'Spring', 2017),  
('83821', 'CS-190', '2', 'Spring', 2017),  
('83821', 'CS-319', '2', 'Spring', 2018),  
('98345', 'EE-181', '1', 'Spring', 2017);  
INSERT INTO student (ID, name, dept\_name, tot\_cred) VALUES  
('00128', 'Zhang', 'Comp. Sci.', 102),  
('12345', 'Shankar', 'Comp. Sci.', 32),  
('19991', 'Brandt', 'History', 80),  
('23121', 'Chavez', 'Finance', 110),  
('44553', 'Peltier', 'Physics', 56),  
('45678', 'Levy', 'Physics', 46),  
('54321', 'Williams', 'Comp. Sci.', 54),  
('55739', 'Sanchez', 'Music', 38),  
('70557', 'Snow', 'Physics', 0),  
('76543', 'Brown', 'Comp. Sci.', 58),  
('76653', 'Aoi', 'Elec. Eng.', 60),  
('98765', 'Bourikas', 'Elec. Eng.', 98),  
('98988', 'Tanaka', 'Biology', 120);  
INSERT INTO takes (ID, course\_id, sec\_id, semester, year, grade) VALUES  
('00128', 'CS-101', 1, 'Fall', 2017, 'A'),  
('00128', 'CS-347', 1, 'Fall', 2017, 'A-'),  
('12345', 'CS-101', 1, 'Fall', 2017, 'C'),  
('12345', 'CS-190', 2, 'Spring', 2017, 'A'),  
('12345', 'CS-315', 1, 'Spring', 2018, 'A'),  
('12345', 'CS-347', 1, 'Fall', 2017, 'A'),  
('19991', 'HIS-351', 1, 'Spring', 2018, 'B'),  
('23121', 'FIN-201', 1, 'Spring', 2018, 'C+'),  
('44553', 'PHY-101', 1, 'Fall', 2017, 'B-'),  
('45678', 'CS-101', 1, 'Fall', 2017, 'F'),  
('45678', 'CS-101', 1, 'Spring', 2018, 'B+'),  
('45678', 'CS-319', 1, 'Spring', 2018, 'B'),  
('54321', 'CS-101', 1, 'Fall', 2017, 'A-'),  
('54321', 'CS-190', 2, 'Spring', 2017, 'B+'),  
('55739', 'MU-199', 1, 'Spring', 2018, 'A-'),  
('76543', 'CS-101', 1, 'Fall', 2017, 'A'),  
('76543', 'CS-319', 2, 'Spring', 2018, 'A'),  
('76653', 'EE-181', 1, 'Spring', 2017, 'C'),  
('98765', 'CS-101', 1, 'Fall', 2017, 'C-'),  
('98765', 'CS-315', 1, 'Spring', 2018, 'B'),  
('98988', 'BIO-101', 1, 'Summer', 2017, 'A'),  
('98988', 'BIO-301', 1, 'Summer', 2018, NULL);  
INSERT INTO advisor (s\_id, i\_id) VALUES  
('00128', '45565'),  
('12345', '10101'),  
('23121', '76543'),  
('44553', '22222'),  
('45678', '22222'),  
('76543', '45565'),  
('76653', '98345'),  
('98765', '98345'),  
('98988', '76766');  
INSERT INTO timeslot (time\_slot\_id, day, start\_time, end\_time) VALUES  
('A', 'M', '08**–**00', '08**–**50'),  
('A', 'W', '08**–**00', '08**–**50'),  
('A', 'F', '08**–**00', '08**–**50'),  
('B', 'M', '09**–**00', '09**–**50'),  
('B', 'W', '09**–**00', '09**–**50'),  
('B', 'F', '09**–**00', '09**–**50'),  
('C', 'M', '11**–**00', '11**–**50'),  
('C', 'W', '11**–**00', '11**–**50'),  
('C', 'F', '11**–**00', '11**–**50'),  
('D', 'M', '13**–**00', '13**–**50'),  
('D', 'W', '13**–**00', '13**–**50'),  
('D', 'F', '13**–**00', '13**–**50'),  
('E', 'T', '10**–**30', '11**–**45'),  
('E', 'R', '10**–**30', '11**–**45'),  
('F', 'T', '14**–**30', '15**–**45'),  
('F', 'R', '14**–**30', '15**–**45'),  
('G', 'M', '16**–**00', '16**–**50'),  
('G', 'W', '16**–**00', '16**–**50'),  
('G', 'F', '16**–**00', '16**–**50'),  
('H', 'W', '10**–**00', '12**–**30');  
INSERT INTO prereq (course\_id, prereq\_id) VALUES  
('BIO-301', 'BIO-101'),  
('BIO-399', 'BIO-101'),  
('CS-190', 'CS-101'),  
('CS-315', 'CS-101'),  
('CS-319', 'CS-101'),  
('CS-347', 'CS-101'),  
('EE-181', 'PHY-101');  
INSERT INTO timeslot\_hr\_min (time\_slot\_id, day, start\_hr, start\_min, end\_hr, end\_min) VALUES  
('A', 'M', 8, 0, 8, 50),  
('A', 'W', 8, 0, 8, 50),  
('A', 'F', 8, 0, 8, 50),  
('B', 'M', 9, 0, 9, 50),  
('B', 'W', 9, 0, 9, 50),  
('B', 'F', 9, 0, 9, 50),  
('C', 'M', 11, 0, 11, 50),  
('C', 'W', 11, 0, 11, 50),  
('C', 'F', 11, 0, 11, 50),  
('D', 'M', 13, 0, 13, 50),  
('D', 'W', 13, 0, 13, 50),  
('D', 'F', 13, 0, 13, 50),  
('E', 'T', 10, 30, 11, 45),  
('E', 'R', 10, 30, 11, 45),  
('F', 'T', 14, 30, 15, 45),  
('F', 'R', 14, 30, 15, 45),  
('G', 'M', 16, 0, 16, 50),  
('G', 'W', 16, 0, 16, 50),  
('G', 'F', 16, 0, 16, 50),  
('H', 'W', 10, 0, 12, 30);

**Output –**

********

****

****

**Conclusion –** In conclusion we can say that we have successfully inserted the data in the database.

**Problem no –** 03

**Problem Name –** Write SQL queries using delete, drop table, alter table command.

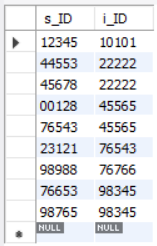
**Query –**

delete from advisor;

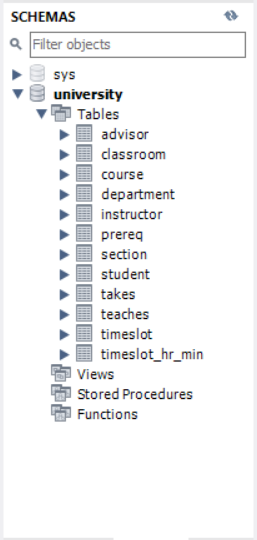
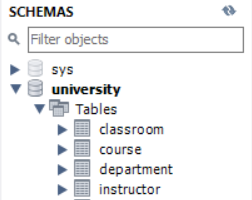
drop table advisor;

alter table advisor add salary numeric(8,0);

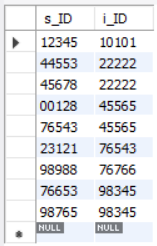
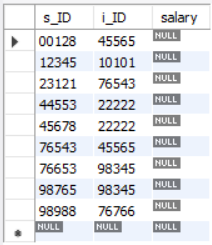
alter table advisor drop salary;

**Output –**

delete from advisor

****

drop table advisor

****

alter table … drop …

alter table … add …

**Conclusion –** These commands are essential for managing data (DELETE), removing tables (DROP TABLE), and modifying table structures (ALTER TABLE) effectively in a database. In conclusion, we can say that we have successfully written SQL queries using delete, drop table, alter table command.

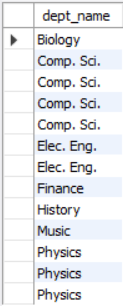
**Problem no –** 04

**Problem Name –** Write a query searching for an attribute.

**Query –**

select dept\_name from student;

select ID,name from student;

**Output –**

**Conclusion –** These queries demonstrate how to retrieve specific attributes (columns) from a table using the SELECT statement. This is a fundamental SQL operation for extracting relevant data from a database.

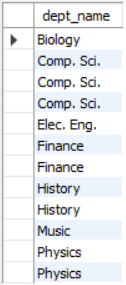
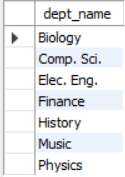
**Problem no –** 05

**Problem Name –** Write queries by implementing the DISTINCT and ALL keywords.

**Query –**

select all dept\_name from instructor;

select distinct dept\_name from instructor;

**Output –**

distinct dept\_name

all dept\_name

**Conclusion –** From the output of the queries we have observed the behaviors of the “**DISTINCT**” and “**ALL”** keywords.

**Problem no –** 06

**Problem Name –** Write queries using arithmetic, logical and relational operators.

**Query** & **Output –**

|  |  |
| --- | --- |
| **Query** | **Output** |
| select ID, name, dept\_name, salary \*1.1 from instructor; |  |
| select name from instructor  where dept\_name = 'Comp. Sci.' and salary > 70000; |  |
| select name, instructor.dept\_name, building  from instructor, department  where instructor.dept\_name= department.dept\_name; |  |
| select name, course\_id from instructor, teaches  where instructor.ID= teaches.ID and instructor.dept\_name = 'Comp. Sci.'; |  |

**Conclusion –** In conclusion, we can say that we have successfully written SQL queries using arithmetic, logical, and relational operators.

**Problem no –** 07

## 07. Write queries using renaming (AS clause) operation.

**Problem name –** Write queries using renaming (AS clause) operation.

**Query** & **Output –**

|  |  |
| --- | --- |
| **Query** | **Output** |
| SELECT DISTINCT T.name FROM instructor as T,  instructor as S WHERE T.salary > S.salary and S.dept\_name = "Biology"; | https://lh7-rt.googleusercontent.com/docsz/AD_4nXerUql0YlPRk1vcQEe87in-OEPo3cBa0nKRlkmS0ov6uTi00E-IHa_U1ZF-eFz0aw-MofZXDNpLW_lr27jBBLMO_YmYM6kOpBOfUsebMn3Z0yQXAKqsoNwdUm5wHyJ5NiU1zr3_XA?key=9AkZjpJyWq4JNBM9J75qb3ad |

**Conclusion –** In this SQL query we can see that the same instructor table is used twice to get every distinct names of the instructors that has more salary than the one who have the least salary. In other word this query is used to get the names of the instructors that do not have the least salary. But in order to get the names we needed to use the same instructor table twice, the touple of name from the T is inserted into the output query every time T’s salary is more that the S’s salary. In conclusion we can say that we have successfully wrote a SQL query using “**AS**” clause.

**Problem no –** 08

## 08. Write queries using BETWEEN keyword and comparison operations.

**Problem name –** Write queries using **BETWEEN** keyword and comparison operations.

**Query** & **Output –**

|  |  |
| --- | --- |
| **Query** | **Output** |
| select name  from instructor where salary between 90000 and 100000; |  |
| select name  from instructor where (salary>= 90000) and (salary<=100000); |  |

**Conclusion–** These queries highlight two methods to filter data based on a range condition for the salary attribute in the instructor table. The first query uses the BETWEEN keyword, a concise and intuitive way to specify a range. The second query employs explicit comparison operators, providing an alternative way to express the same logic. So therefore, we can say that we have successfully written queries using BETWEEN keyword and comparison operations.

**Problem no –** 09

## 09. Write queries using aggregate functions. (AVG, MAX, MIN, SUM, COUNT)

**Problem name –** Write queries using aggregate functions. (AVG, MAX, MIN, SUM, COUNT)

**Query & Output –**

|  |  |
| --- | --- |
| **Query** | **Output** |
| select avg (salary) from instructor where dept\_name= 'Comp. Sci.'; |  |
| select dept\_name, count(ID) from instructor group by dept\_name; |  |
| select sum(salary) from instructor; |  |
| select max(salary) from instructor; |  |
| select min(salary) from instructor; |  |

**Conclusion –** These queries showcase using SQL aggregate functions, such as AVG, COUNT, SUM, MAX, and MIN, to perform calculations on groups of rows. Each function serves a distinct purpose, from calculating averages to determining the total, maximum, or minimum values, or counting rows. Thus, we can say that we have successfully written queries using aggregate functions. (AVG, MAX, MIN, SUM, COUNT).

**Problem no –** 10

## 10. Write subqueries for fetching specific data and show the usages of SOME and ALL clauses before the subqueries.

**Problem name –** Write subqueries for fetching specific data and show the usages of SOME and ALL clauses before the subqueries.

**Query & Output –**

|  |  |
| --- | --- |
| **Query** | **Output** |
| SELECT instructor.name FROM instructor WHERE instructor.salary > some(SELECT instructor.salary FROM instructor WHERE instructor.dept\_name = 'Biology'); |  |
| SELECT instructor.name FROM instructor WHERE instructor.salary < all(SELECT instructor.salary FROM instructor WHERE instructor.dept\_name = 'Biology'); |  |

**Conclusion –** In the first query one of them gives all the name that has more salary than any of the instructors in the biology department using the “**SOME**” and in the second query using the “**ALL**” we find all the names of the instructors that less than all of the instructors in the biology department.

**Problem no –** 11

## 11. Write queries using string operations, attribute specification, and ORDER BY clause.

**Problem name –** Write queries using string operations, attribute specification, and ORDER BY clause.

**Query & Output–**

|  |  |
| --- | --- |
| **Query** | **Output** |
| SELECT dept\_name FROM department WHERE building LIKE '%Taylor%'; |  |
| SELECT instructor.\* FROM instructor, teaches WHERE instructor.ID = teaches.ID; |  |
| SELECT name FROM instructor WHERE dept\_name = 'Comp. Sci.' ORDER BY name; |  |
| SELECT \* FROM instructor ORDER BY salary DESC; |  |

**Conclusion –** In conclusion, we can say that we have successfully have written the SQL query to do string matching operation and the sorting operation using “**ORDER BY**”.

**Problem no –** 12

## 12. Write queries using set operations (UNION, INTERSECT, etc.).

**Problem name –** Write queries using set operations (UNION, INTERSECT, EXCEPT).

**Query & Output –**

|  |  |
| --- | --- |
| **Query** | **Output** |
| (select course\_id from section where semester = 'Fall' and year= 2017) union (select course\_id from section where semester = 'Spring' and year= 2018); |  |
| (select course\_id from section where semester = 'Fall' and year= 2017) intersect (select course\_id from section where semester = 'Spring' and year= 2018); |  |
| (select course\_id from section where semester = 'Fall' and year= 2017) except (select course\_id from section where semester = 'Spring' and year= 2018); |  |

**Conclusion –** With the outputs and comparisons from the datasets we can say that we have successfully written queries using set operations (UNION, INTERSECT, EXCEPT).

**Problem no –** 13

## 13. Write queries using set membership, set comparison, and testing for empty relationships.

**Problem name –** Write queries using set membership, set comparison, and testing for empty relationships.

**Query & Output –**

|  |  |
| --- | --- |
| **Query** | **Output** |
| select distinct course\_id from section where semester = 'Fall' and year= 2017 and course\_id in (  select course\_id   from section   where semester = 'Spring' and year= 2018 ); |  |
| select distinct course\_id from section where semester = 'Fall' and year= 2017 and course\_id not in (  select course\_id  from section  where semester = 'Spring' and year= 2018 ); |  |
| select distinct T.name from instructor as T, instructor as S where T.salary > S.salary and S.dept\_name = 'Biology'; |  |
| select name from instructor where salary > some (select salary from instructor where dept\_name = 'Biology'); |  |
| select name  from instructor  where salary > all (  select salary  from instructor  where dept\_name = 'Biology'  ); |  |
| select course\_id  from section as S  where semester = 'Fall' and year = 2017 and exists(  select \*   from section as T   where semester = 'Spring' and year= 2018 and S.course\_id = T.course\_id ); |  |

**Conclusion –** In conclusion we can say that we have successfully wrote queries to use set membership, set comparison, and testing for empty relationships using “**IN**”, “**SOME**”, “**ALL**” and “**EXISTS**” clauses.

**Problem no –** 14

## 14. Write queries on multiple relations and the use of NATURAL JOIN keyword

**Problem name –** Write queries on multiple relations and the use of NATURAL JOIN keyword.

**Query & Output –**

|  |  |
| --- | --- |
| **Query** | **Output** |
| select name, course\_id from student natural join takes; |  |
| select name, title from student natural join takes, course where takes.course\_id = course.course\_id; |  |

**Conclusion –** In conclusion we can say that we have successfully written queries on multiple relations with the use of NATURAL JOIN keyword.

**Problem no –** 15

## 15. Write queries using different types of joins (INNER JOIN, LEFT JOIN, RIGHT JOIN).

**Problem name –** Write queries using different types of joins (INNER JOIN, LEFT JOIN, RIGHT JOIN).

**Query & Output –**

|  |  |  |  |
| --- | --- | --- | --- |
| **Query** | | **Output** | |
| select \* from student join takes  on student.ID = takes.ID; | |  |
| select \*  from course  natural left outer join prereq; | |  |
| select \*  from course natural right outer join prereq; | |  |

**Conclusion –** In the command we can observe that simple using “**JOIN**” command using “**ON**” keyword returns an inner join and LEFT or RIGHT outer join requires the usage of explicit command. In conclusion, we can say that we have successfully wrote SQL commands using inner join, left join and right join.

**Problem no –** 16

## 16. Write SQL queries to create and manipulate views for displaying student details with their respective course names and grades.

**Problem name –** Write SQL queries to create and manipulate views for displaying student details with their respective course names and grades.

**Query and Output –**

|  |  |
| --- | --- |
| **Query** | **Output** |
| create view student\_grades as select  student.ID as ID, student.name as name, student.dept\_name as dept\_name, student.tot\_cred as tot\_cred, takes.course\_id as course\_id, takes.grade as grade from student, takes  where student.ID = takes.ID; |  |
| select \* from student\_grades where dept\_name='Comp. Sci.'; |  |
| update takes set grade='A' where ID = 128 and course\_id = 'CS-347'; |  |
| drop view student\_grades; |  |

**Conclusion –** In the query we cannot implement the view to modify the grade of Zang to ‘A’. Usually these kinds of operation are why view is used. But a database admin can easily have access to the takes table. So, this gives an added layer of security to the data. In conclusion we can say that we have successfully have written SQL queries to create and manipulate views for displaying student details with their respective course names and grades.

**Problem no –** 17

## 17. Write SQL queries to create a trigger that automatically updates the budget of a department in the department table whenever an instructor's salary is updated in the instructor table.

**Problem name –** Write SQL queries to create a trigger that automatically updates the budget of a department in the department table whenever an instructor's salary is updated in the instructor table.

**Query & Output –**

|  |  |
| --- | --- |
| **Query** | **Output** |
| DELIMITER $$ CREATE TRIGGER adjust\_department\_budget  AFTER INSERT ON instructor  FOR EACH ROW  BEGIN IF (NEW.salary IS NOT NULL) THEN UPDATE department  SET budget = budget + (NEW.salary/1000) WHERE dept\_name = NEW.dept\_name;   END IF;  END;  DELIMITER; |  |
| DELIMITER $$ CREATE TRIGGER adjust\_department\_budget2 AFTER UPDATE ON instructor FOR EACH ROW BEGIN IF (NEW.salary IS NOT NULL AND OLD.salary IS NOT NULL) THEN UPDATE department SET budget = budget + (NEW.salary - OLD.salary)/1000 WHERE dept\_name = NEW.dept\_name;  end if; IF NEW.salary IS NOT NULL and OLD.salary IS NULL THEN update department set budget = budget + new.salary/1000 where dept\_name = new.dept\_name; END IF; END; DELIMITER; |  |
| update instructor set salary = salary\*1.10; select \* from department; |  |

**Conclusion –** In conclusion we can say that we have successfully wrote a trigger in SQL that automatically updates the budget of a department in the department table whenever an instructor's salary is updated in the instructor table or a new instructor in hired in the department.

**Problem no –** 18

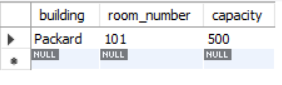
## 18. Write SQL queries to create an index on the room\_number column of the classroom table to speed up searches for classrooms based on their room numbers. Additionally, demonstrate how to drop this index if it is no longer needed.

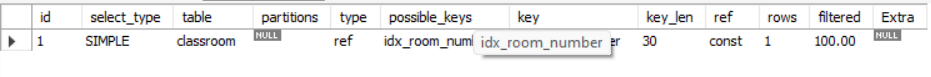
**Problem name –** Write SQL queries to create an index on the room\_number column of the classroom table to speed up searches for classrooms based on their room numbers. Additionally, demonstrate how to drop this index if it is no longer needed.

**Query –**

CREATE INDEX idx\_room\_number  
ON classroom(room\_number);  
  
SELECT building, room\_number, capacity  
FROM classroom  
WHERE room\_number = '101';  
  
-- to see wheather database is using index yes or not  
EXPLAIN SELECT building, room\_number, capacity  
FROM classroom  
WHERE room\_number = '101';  
  
DROP INDEX idx\_room\_number  
ON classroom(room\_number);

**Output–**





**Conclusion –** Indexes significantly enhance performance for queries involving search and filter operations by reducing the time required to locate data. They are essential for improving read performance in large datasets but should be used judiciously due to the additional storage and maintenance overhead they introduce. Regular monitoring and optimization of indexes are crucial for maintaining a balanced database system.