University Database

Various software programs employ a database management system, or DBMS, as a tool to arrange and retrieve related data. The University Database compiles vital information regarding departments, classes, instructors, students, classrooms, and other elements. It is arranged according to user requirements, with a focus on key organizational elements:

- ✓ Departments are uniquely identified by dept_name, located in particular buildings, and each with a designated budget.
- ✓ Courses belonging to departments contain details like course_id, title, dept_name, credits, and any prerequisites that may apply.
- ✓ Instructors have unique IDs, with details on their names, affiliated departments (dept_name), and salaries.
- ✓ Students, each identified by a unique ID, have names, major departments (dept_name), and total credit hours earned (tot_cred).
- ✓ Classrooms are specified by the building name, room number, and meeting schedule (time_slot_id).
- ✓ Class sections are identified by course_id, sec_id, year, and semester, and are linked to a building, room number, and time slot (time_slot_id).
- ✓ Teaching assignments allocate instructors to specific sections within departments.
- ▪ Student course enrollments record the courses and sections in which students are registered.

The University Database model, which includes a schema, Entity-Relationship (E-R) diagram, SQL definitions, and sample entries, creates a comprehensive system for organizing and retrieving university-related data.

Full Schema

Schema for a University Database:

- **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`)

ER diagram

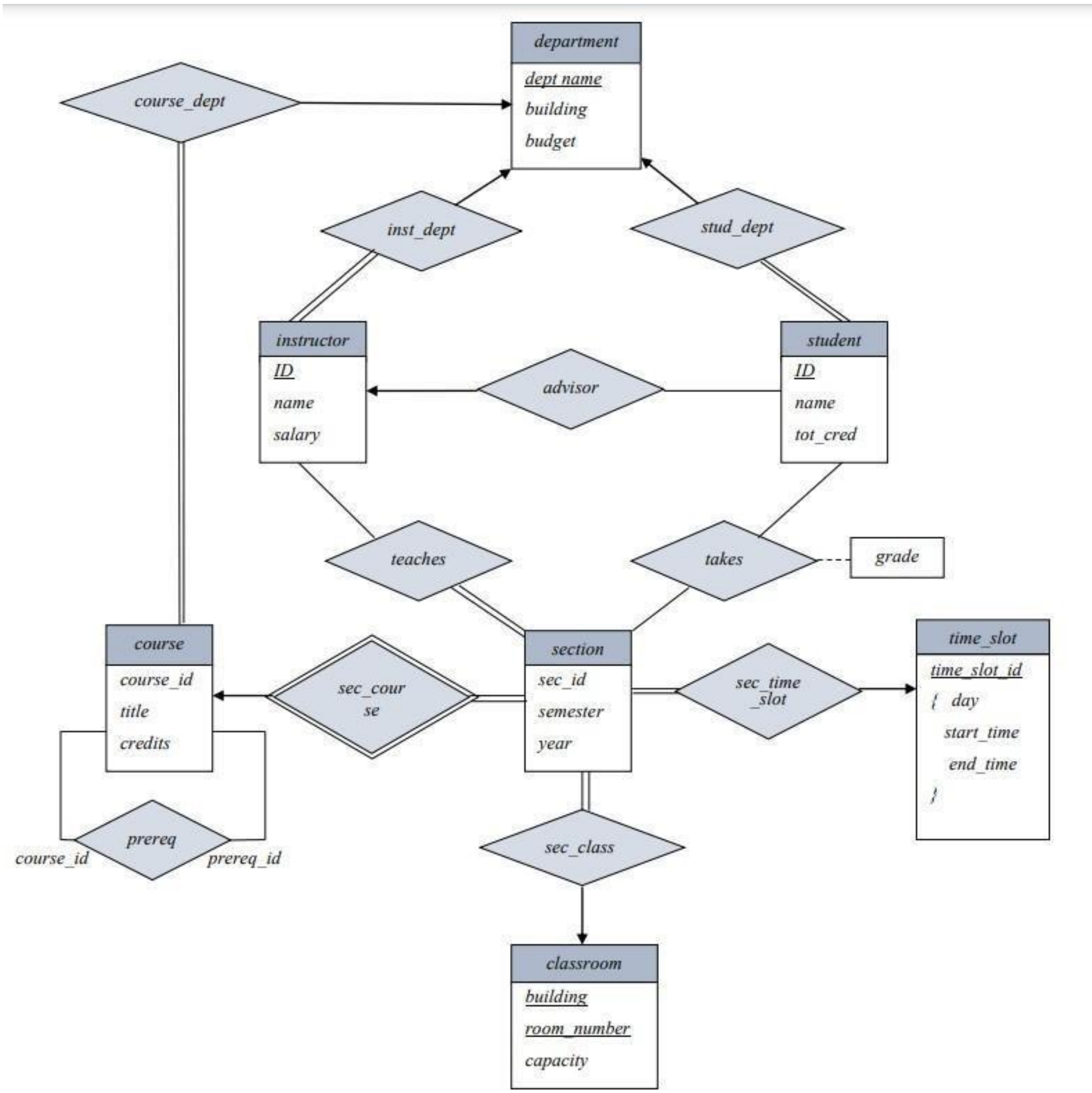

Fig: E-R diagram for a university

Schema Diagram for University Database:

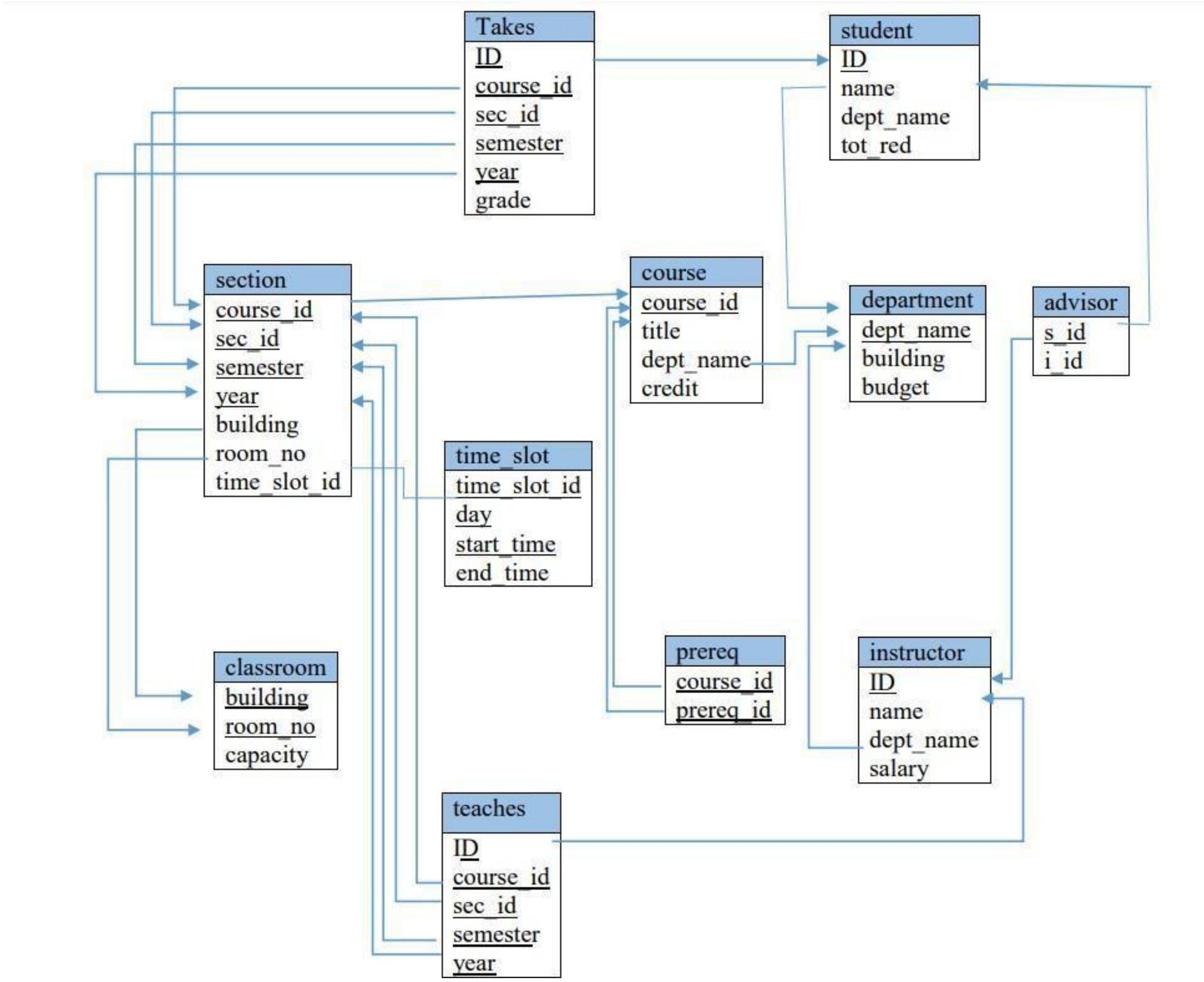

Fig: Schema diagram for a university enterprise

Experiment No: 01

Experiment Name: Write SQL queries using integrity constraints to create tables for a database.

Introduction:
Data integrity is the cornerstone of a trustworthy database. Integrity restrictions are crucial for maintaining consistency and implementing company standards since they act as vigilant guardians to ensure that data remains accurate and legitimate. This topic examines developing SQL queries that consider these constraints in order to provide robust and dependable tables that support a well-structured database.

Program Code:

```
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),
        primary key
        (dept_name));

create table course(
   course_id varchar (7),
        title varchar (50),
        dept_name varchar (20),
        credits numeric (2,0),
        primary key (course_id));

create table instructor(
   ID varchar (5),
        name varchar (20) not null,
        dept_name varchar (20),
        salary numeric (8,2),
        primary key (ID));

create table section(
   course_id varchar (8),
```

```
        sec_id varchar (8),
        semester varchar (6),
        year1 numeric (4,0),
        building varchar (15),
        room_number varchar (7),
        time_slot_id varchar (4),
        primary key (course_id, sec_id, semester, year1));

create table teaches(
   ID varchar (5),
        course_id varchar (8),
        sec_id varchar (8),
        semester varchar (6),
        year1 numeric (4,0),
        primary key (ID, course_id, sec_id, semester, year1));

  create table student(
        ID      varchar (5),
        name varchar (20),
        dept_name varchar (20),
        tot_cred numeric (3,0),
        primary key (ID));

create table takes(
   ID varchar (5),
        course_id varchar (8),
        sec_id varchar (8),
        semester varchar (6),
        year1 numeric (4,0),
        grade varchar (2),
        primary key (ID, course_id, sec_id, semester, year1));

create table advisor(
      s_ID varchar
      (5),
        i_ID varchar (5),
        primary key
        (s_ID));

create table prereq(
   course_id
   varchar(8),
        prereq_id varchar(8),
        primary key (course_id, prereq_id));
```

```
create table timeslot(
   time_slot_id varchar
   (4),
        day varchar (1),
        start_hr numeric (2),
        start_min numeric (2),
        end_hr numeric (2),
        end_min numeric (2),
        primary key (time_slot_id, day, start_hr, start_min));
```

Output:

Integrity constraints are used to create tables for university database.

Discussion:

We successfully imposed integrity constraints and created the required tables using SQL in XAMPP. The CREATE TABLE statement was used to create the database structure, and integrity constraints ensure data consistency and accuracy.

Experiment No: 02

Experiment Name: Write SQL queries to insert values into tables in the university database.

Introduction:

SQL's INSERT INTO statement is commonly used to add new records to a table. You can either include the names of the columns and their values or omit them if the values match the order specified in the table definition. Use the INSERT INTO SELECT command to populate data from another table or query result, or use a comma-separated list of values to insert many items at once.

Program Code:

```
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 VALUES("Biology","Watson",90000);
INSERT INTO department
VALUES("Comp.Sci.","Taylor",100000); INSERT INTO
department VALUES("Elec.Eng.","Taylor",85000); INSERT INTO
department VALUES("Finance","Painter",120000); INSERT INTO
department VALUES("History","Painter",50000); INSERT INTO
department VALUES("Music","Packrad",80000); INSERT INTO
department VALUES("Physics","Watson",70000);

INSERT INTO course VALUES("BIO-101","Intro. to
Biology","Biology",4); INSERT INTO course VALUES("BIO-
301","Genetics","Biology",4);
INSERT INTO course VALUES("BIO-399","Computational Biology","Biology",3);
INSERT INTO course VALUES("CS-101","Intro. to Computer Science", "Comp.
Sci.",4);
INSERT INTO course VALUES("CS-109","Game Design","Comp. Sci.",
4); INSERT INTO course VALUES("CS-315","Robotics", "Comp. Sci.", 3);
INSERT INTO course VALUES("CS-319","Image Processing", "Comp. Sci.", 3);
INSERT INTO course VALUES("CS-347","Databse System Concepts", "Comp. Sci.",
3);
INSERT INTO course VALUES("EE-181","Intro. to Digital Systems","Elec.
Eng",3); INSERT INTO course VALUES("FIN-201","Investment
Banking","Finance", 3); INSERT INTO course VALUES("HIS-351","World
History","History", 3);
INSERT INTO course VALUES("MU-199","Music Video Production", "Music", 3);
```

```
INSERT INTO course VALUES("PHY-101","Physical Principles", "Physics", 4);

INSERT INTO instructor VALUES(10101,"Srinivasan","Comp.
Sci.",65000); INSERT INTO instructor VALUES(45565,"Katz", "Comp.
Sci.", 75000); INSERT INTO instructor VALUES(83821,"Brandt", "Comp.
Sci.",92000); INSERT INTO instructor
VALUES(12121,"Wu","Finance",90000);
INSERT INTO instructor VALUES(76543,"Singh","Finance",80000);
INSERT INTO instructor VALUES(15151,"Mozart","Music",40000);
INSERT INTO instructor
VALUES(22222,"Einstein","Physics",95000); INSERT INTO
instructor VALUES(33456,"Gold","Physics",87000); INSERT INTO
instructor VALUES(32343,"El Said","History",60000); INSERT INTO
instructor VALUES(58583,"Califieri","History",62000); INSERT INTO
instructor VALUES(76766,"Crick","Biology",72000); INSERT INTO
instructor VALUES(98345,"Kin","Elec. Eng",80000);

INSERT INTO section VALUES ("BIO-101", 1, "Summer", 2017, "Painter",514,
"B"); INSERT INTO section VALUES ("BIO-301", 1, "Summer", 2018,
"Painter",514, "A"); INSERT INTO section VALUES ("CS-101", 1, "Fall", 2017,
"Packard", 101, "H"); INSERT INTO section VALUES ("CS-101", 1, "Spring", 2018,
"Packard", 101, "F"); INSERT INTO section VALUES ("CS-190", 1, "Spring", 2017,
"Taylor", 3128, "E"); INSERT INTO section VALUES ("CS-190", 2, "Spring", 2017,
"Taylor", 3128, "A"); INSERT INTO section VALUES ("CS-315", 1, "Spring", 2018,
"Watson", 120, "D"); INSERT INTO section VALUES ("CS-319", 1, "Spring", 2018,
"Watson", 100, "B"); INSERT INTO section VALUES ("CS-319", 2, "Spring", 2018,
"Taylor", 3128, "C"); INSERT INTO section VALUES ("CS-347", 1, "Fall", 2017,
"Taylor", 3128, "A"); INSERT INTO section VALUES ("EE-181", 1, "Spring", 2017,
"Taylor", 3128, "C"); INSERT INTO section VALUES ("FIN-201", 1, "Spring", 2018,
"Packard",101, "B"); INSERT INTO section VALUES ("HIS-351", 1, "Spring", 2018,
"Painter",514, "C"); INSERT INTO section VALUES ("MU-199", 1, "Spring", 2018,
"Packard", 101, "D"); INSERT INTO section VALUES ("PHY-101", 1, "Fall", 2017,
"Watson", 100, "A");

INSERT INTO teaches (ID, course_id, sec_id, semester,
year1) 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, year1,
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, 'Fall', 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 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 (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:

building	room_nun	capacity
Packard	101	500
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

course_id	sec_id	semester	year	building	room_nun	time_slot_id
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

Classroom:

ID	course_id	sec_id	semester	year	grade
128	CS-101	1	Fall	2017	A
128	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

ID	name	dept_nam	salary
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

dept_nam	building	budget
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

ID	course_id	sec_id	semester	year
76766	BIO-101	1	Summer	2017
76766	BIO-301	1	Summer	2018
10101	CS-101	1	Fall	2017
45565	CS-101	1	Spring	2018
83821	CS-190	1	Spring	2017
83821	CS-190	2	Spring	2017
10101	CS-315	1	Spring	2018
45565	CS-319	1	Spring	2018
83821	CS-319	2	Spring	2018
10101	CS-347	1	Fall	2017
98345	EE-181	1	Spring	2017
12121	FIN-201	1	Spring	2018
32343	HIS-351	1	Spring	2018
15151	MU-199	1	Spring	2018
22222	PHY-101	1	Fall	2017

time_slot_	day	start_hr	start_min	end_hr	end_min
A	F	8	0	8	50
A	M	8	0	8	50
A	W	8	0	8	50
B	F	9	0	9	50
B	M	9	0	9	50
B	W	9	0	9	50
C	F	11	0	11	50
C	M	11	0	11	50
C	W	11	0	11	50
D	F	13	0	13	50
D	M	13	0	13	50
D	W	13	0	13	50
E	R	10	30	11	45
E	T	10	30	11	45
F	R	14	30	15	45
F	T	14	30	15	45
G	F	16	0	16	50
G	M	16	0	16	50
G	W	16	0	16	50
H	W	10	0	12	30

ID	name	dept_nam	tot_cred
128	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

course_id	title	dept_nam	credits
BIO-101	Intro. to B	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computati	Biology	3
CS-101	Intro. to C	Comp. Sci.	4
CS-190	Game Desi	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Pro	Comp. Sci.	3
CS-347	Database !	Comp. Sci.	3
EE-181	Intro. to D	Elec. Eng.	3
FIN-201	Investmen	Finance	3
HIS-351	World Hist	History	3
MU-199	Music Vide	Music	3
PHY-101	Physical Pr	Physics	4

s_ID	i_ID
12345	10101
44553	22222
45678	22222
128	45565
76543	45565
23121	76543
98988	76766
76653	98345
98765	98345

course_id	prereq_id
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

Discussion:

With XAMPP and SQL, we were able to successfully enter data into tables. To add new records to the student database table, the INSERT INTO statement was used, defining the target columns and the values that went with them.

Experiment No: 03

Experiment Name: Write SQL query using insert and delete, drop table, alter table command.

Introduction:

SQL provides several commands for managing database tables:

- INSERT: Adds new records to a table, specifying column names and values or relying on the table's column order.
- DELETE: Removes existing records, optionally filtering by a specific condition.
- DROP TABLE: Deletes an entire table and its data. Use with caution.
- ALTER TABLE: Modifies table structure by adding, removing, or changing columns and constraints.

Program Code:

```
INSERT INTO instructor VALUES(10211, 'Smith', 'Biology',
66000); DELETE FROM student;
DROP TABLE teaches;
ALTER TABLE student ADD Height NUMERIC
(2,0); ALTER TABLE student DROP height;
```

Output:

1 row inserted. (Query took 0.0066 seconds.)

```
INSERT
INTO
```

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0010 seconds.)

```
DROP TABLE teaches;
```

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0009 seconds.)

```
ALTER TABLE student ADD Height NUMERIC (2,0);
```

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0010 seconds.)

```
ALTER TABLE student DROP height;
```

Discussion:

We were able to manipulate the student database using the INSERT INTO, DELETE, DROP TABLE, and ALTER TABLE commands in the XAMPP environment. New records were added using the INSERT INTO command, while old entries were deleted using the DELETE statement. The ALTER TABLE command changed the structure of the table, while the DROP TABLE command completely erased the entire table.

Experiment No: 04

Experiment Name:

Write a query for searching for an attribute.

Introduction:

To retrieve specific data from a database table, we utilize SQL's SELECT command. By providing the table name and the required columns, we may extract and analyze the required data.

Program Code:

```
SELECT name
FROM student;

SELECT dept_name
FROM instructor;
```

Output:

name
Zhang
Shankar
Brandt
Chavez
Peltier
Levy
Williams
Sanchez
Snow
Brown
Aoi
Bourikas
Tanaka

dept_name
Biology
Comp. Sci.
Comp. Sci.
Comp. Sci.
Elec. Eng.
Finance
Finance
History
History
Music
Physics
Physics

Discussion:

To facilitate data exploration and analysis, the first query retrieves names from the student table, while the second query retrieves department names from the instructor table.

Experiment No: 05

Experiment Name:
Write queries by implementing the ‘distinct’ and ‘all’ord.

Introduction:
To extract unique or all values from a database table, SQL queries employ the DISTINCT and ALL keywords. These keywords are crucial for tailoring search results to specific needs. While DISTINCT removes duplicate values, ALL offers flexibility in data retrieval and analysis by including all values.

Program Code:
```
SELECT DISTINCT dept_name
FROM instructor;

SELECT ALL dept_name
FROM instructor;
```

Output:

dept_name
Biology
Comp. Sci.
Elec. Eng.
Finance
History
Music
Physics

Discussion:
'DISTINCT' and 'ALL' were employed in SQL queries within the XAMPP. "DISTINCT" contributed to the creation of an understandable and succinct result set by eliminating redundant data. The flexibility of these keywords was demonstrated by the fact that 'ALL' allowed the retrieval of all values. This XAMPP implementation demonstrates how accurate and flexible SQL queries are for modifying data retrieval.

Experiment No: 06

Experiment Name:
Write queries using arithmetic, logical and relational operator.

Instruction:

The creation of SQL queries that utilize relational, logical, and arithmetic operators is examined in this study. Through the creation of logical conditions, calculations, and connections between data, these operators enable users to get data from relational databases accurately and dynamically.

Program Code:

```
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.';
```

Output

ID	name	dept_name	salary * 1.1
10101	Srinivasan	Comp. Sci.	71500
12121	Wu	Finance	99000
15151	Mozart	Music	44000
22222	Einstein	Physics	104500
32343	El Said	History	66000
33456	Gold	Physics	95700
45565	Katz	Comp. Sci.	82500
58583	Califieri	History	68200
76543	Singh	Finance	88000
76766	Crick	Biology	79200
83821	Brandt	Comp. Sci.	101200
98345	Kim	Elec. Eng.	88000

name	course_id
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Katz	CS-101
Katz	CS-319
Brandt	CS-190
Brandt	CS-190
Brandt	CS-319

name
Katz
Brandt

name	dept_nam	building
Crick	Biology	Watson
Srinivasan	Comp. Sci.	Taylor
Katz	Comp. Sci.	Taylor
Brandt	Comp. Sci.	Taylor
Kim	Elec. Eng.	Taylor
Wu	Finance	Painter
Singh	Finance	Painter
El Said	History	Painter
Califieri	History	Painter
Mozart	Music	Packard
Einstein	Physics	Watson
Gold	Physics	Watson

Discussion:
With the aid of XAMPP software, we have effectively used relational, logical, and arithmetic operators with SQL.

Experiment No: 07

Experiment Name: Write queries using renaming (‘as’ clause) operation.

Introduction:

SQL's AS clause allows for the renaming of columns or tables inside queries, improving readability and enabling more natural data interpretation. Custom aliases allow users to simplify and make complex searches easier to understand.

Program Code:

```
SELECT name, course_id
FROM instructor, teaches
WHERE instructor.ID = teaches.ID;

SELECT name AS instructor_name, course_id
FROM instructor, teaches
WHERE instructor.ID = teaches.ID;
SELECT T.name, S.course_id
FROM instructor AS T, teaches AS S WHERE T.ID = S.ID;
SELECT DISTINCT T.name FROM instructor AS T, instructor AS
S WHERE T.salary > S.salary AND S.dept_name = 'Biology';
```

Output:

name	course_id
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Wu	FIN-201
Mozart	MU-199
Einstein	PHY-101
El Said	HIS-351
Katz	CS-101
Katz	CS-319
Crick	BIO-101
Crick	BIO-301
Brandt	CS-190
Brandt	CS-190
Brandt	CS-319
Kim	EE-181

name
Wu
Einstein
Gold
Katz
Singh
Brandt
Kim

name	course_id
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Wu	FIN-201
Mozart	MU-199
Einstein	PHY-101
El Said	HIS-351
Katz	CS-101
Katz	CS-319
Crick	BIO-101
Crick	BIO-301
Brandt	CS-190
Brandt	CS-190
Brandt	CS-319
Kim	EE-181

instructor_	course_id
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Wu	FIN-201
Mozart	MU-199
Einstein	PHY-101
El Said	HIS-351
Katz	CS-101
Katz	CS-319
Crick	BIO-101
Crick	BIO-301
Brandt	CS-190
Brandt	CS-190
Brandt	CS-319
Kim	EE-181

Discussion:

We successfully implemented the AS clause in SQL using XAMPP. Tables and fields can be renamed inside of queries thanks to this SQL functionality. We demonstrated the AS clause's usefulness in altering field and table names for better readability and comprehension by using it to make our queries more clear and structured.

Experiment No: 08

Experiment Name:Write queries using 'between' keyword and comparison operation.

Instruction:
We can use SQL's BETWEEN keyword to filter data inside a given range. This keyword enables us to define a certain column's lower and upper bounds.

Program Code:
```
SELECT name
FROM instructor
WHERE salary BETWEEN 90000 AND 100000;

SELECT name FROM instructor
WHERE salary <= 100000 AND salary >= 90000;

SELECT name, course_id
FROM instructor, teaches
WHERE instructor.ID = teaches.ID AND dept_name = 'Biology';

SELECT name, course_id
FROM instructor, teaches
WHERE (instructor.ID, dept_name) =(teaches.ID, 'Biology');
```

Output:

name
Wu
Einstein
Brandt

name
Wu
Einstein
Brandt

name	course_id
Crick	BIO-101
Crick	BIO-301

name	course_id
Crick	BIO-101
Crick	BIO-301

Discussion:
In the XAMPP environment, we were able to effectively leverage comparison operators and SQL's BETWEEN keyword to enhance data retrieval. By establishing ranges and constraints, we were able to obtain precise data that met our specific needs. The combination of SQL and XAMPP enhances our ability to rapidly get relevant data.

Experiment No: 09

Experiment Name:

Write queries using Aggregate function.

Introduction:

SQL's aggregate functions offer strong capabilities for data analysis and summarization. These functions give us useful information about the data by enabling us to do calculations on sets of rows. We can better grasp our data and arrive at wise decisions by employing aggregate functions.

Program Code:

```
select avg (salary) from instructor
where dept_name= "Comp. Sci.";

select avg (salary) as avg_salary from instructor
where dept_name= "Comp. Sci.";

select count(*) from course;
select dept_name, avg (salary) as avg_salary from instructor group by dept_name;
select dept_name, count(distinct ID) as instr_count
from instructor natural join teaches
where semester = "Spring" and year = 2010 group by dept_name;

select dept_name, avg (salary) as avg_salary
from instructor group by dept_name having avg (salary) > 42000;

select course_id, semester, year, sec_id, avg (tot_cred)
from takes natural join student where year = 2009 group by course_id,
semester, year, sec_id having count(ID) >= 2;
```

Output:

count(*)
13

avg (salary)
77333.3333

avg (salary)
77333.3333

dept_name	avg_salary	dept_name	avg_salary
Biology	72000	Biology	72000
Comp. Sci.	77333.33	Comp. Sci.	77333.33
Elec. Eng.	80000	Elec. Eng.	80000
Finance	85000	Finance	85000
History	61000	History	61000
Music	40000	Music	40000
Physics	91000	Physics	91000

dept_name	instr_count

course_id	semester	year	sec_id	avg (tot_cred)

Discussion:

We've used the Aggregate function successfully. use XAMPP software in conjunction with SQL.

Experiment No: 10

Experiment Name:

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

Introduction:

inquiries nestled inside other inquiries are called subqueries. They are employed to obtain information that can be supplied to the primary inquiry. By permitting comparisons with a collection of results, the SOME and ALL clauses increase the utility of subqueries. The use of subqueries with these clauses is illustrated in this report.

Program Code:

```
SELECT name
FROM student
WHERE tot_cred > (
   SELECT tot_cred
   FROM student
   WHERE ID = '12345'
);

SELECT course_id, title
FROM course
WHERE credits > SOME (
   SELECT credits
   FROM course
   WHERE dept_name = 'Comp. Sci.'
);

SELECT name
FROM instructor
WHERE salary > ALL (
   SELECT salary
   FROM instructor
   WHERE dept_name = 'History'
);

SELECT dept_name
FROM department
WHERE budget < SOME (
```

```
    SELECT budget
    FROM department
    WHERE dept_name IN ('Physics', 'Comp. Sci.')
);
```

Output:

		name
Copy	Delete	Zhang
Copy	Delete	Brandt
Copy	Delete	Chavez
Copy	Delete	Peltier
Copy	Delete	Levy
Copy	Delete	Williams
Copy	Delete	Sanchez
Copy	Delete	Brown
Copy	Delete	Aoi
Copy	Delete	Bourikas
Copy	Delete	Tanaka

course_id	title
BIO-101	Intro. to Biology
BIO-301	Genetics
CS-101	Intro. to Computer Science
CS-109	Game Design
PHY-101	Physical Principles

		dept_name
Copy	Delete	History

		name
Copy	Delete	Srinivasan
Copy	Delete	Wu
Copy	Delete	Einstein
Copy	Delete	Gold
Copy	Delete	Katz
Copy	Delete	Singh
Copy	Delete	Crick
Copy	Delete	Brandt
Copy	Delete	Kin

Discussion:

Subqueries embed one query inside another, enabling dynamic data retrieval. Subqueries are improved by the SOME and ALL clauses, which allow conditional comparisons across a range of values.

SOME Clause: Matches conditions for at least one value in the result set, offering flexibility when dealing with partial matches.

ALL Clause: Enforces conditions to hold true for all values in the result set, suitable for strict comparisons.

These constructs simplify complex filtering logic, making queries more readable and efficient. Careful use of indexing and optimization techniques is necessary to avoid performance bottlenecks in subqueries.

Experiment No: 11

Experiment Name:

Write queries using String operation, attribute specification and ‘order by’ clause.

Instruction:

Our goal is to create SQL queries that sort results, pick particular properties, and manipulate strings. This entails combining the SELECT and ORDER BY clauses with string functions like CONCAT and SUBSTRING.

Program Code:

```
SELECT dept_name FROM department
WHERE building LIKE '%Watson%';

SELECT instructor.*
FROM instructor,
teaches
WHERE instructor.ID = teaches.ID;

SELECT name
FROM
instructor
WHERE dept_name = 'Physics'
ORDER BY name;
SELECT *
FROM instructor
ORDER BY salary DESC, name ASC;
```

Output:

dept_name
Biology
Physics

name
Einstein
Gold

ID	name	dept_name	salary
22222	Einstein	Physics	95000
83821	Brandt	Comp. Sci.	92000
12121	Wu	Finance	90000
33456	Gold	Physics	87000
98345	Kim	Elec. Eng.	80000
76543	Singh	Finance	80000
45565	Katz	Comp. Sci.	75000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
32343	El Said	History	60000
15151	Mozart	Music	40000

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
10101	Srinivasan	Comp. Sci.	65000
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
45565	Katz	Comp. Sci.	75000
76766	Crick	Biology	72000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
83821	Brandt	Comp. Sci.	92000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Discussion:

By expertly utilizing SQL's String operations, attribute specification, and 'ORDER BY' clause, we were able to optimize searches using XAMPP. Through string manipulation, attribute specification, and result organization utilizing functions like CONCAT to enhance data display, we showcased the versatility of SQL in conjunction with XAMPP.

Experiment No: 12

Experiment Name:

Write queries using Set operation.

Introduction:

Create SQL queries that mix or compare data from several tables using set operations like UNION or INTERSECT. To combine different values, for example, use SELECT column FROM table1 UNION SELECT column FROM table2.

Program Code:

```
SELECT course_id
FROM section
WHERE semester = 'Fall' AND YEAR = 2017;

SELECT course_id
FROM section
WHERE semester = 'Spring' AND year = 2018;
```

Union operation:

```
(SELECT course_id
FROM section
WHERE semester = "Fall" AND year= 2017)
UNION
(SELECT course_id
FROM section
WHERE semester = "Spring" AND year= 2018);
```

Intersect operation:

```
(SELECT course_id
FROM section
WHERE semester = "Fall" AND year= 2009)
INTERSECT
(SELECT course_id
FROM section
WHERE semester = "Spring" AND year= 2010);
```

Except operation:

```
(SELECT course_id
FROM section
WHERE semester = "Fall" AND year= 2009)
EXCEPT
(SELECT course_id
FROM section WHERE semester = "Spring" AND year= 2010);
```

Output:

course_id
CS-101
CS-347
PHY-101

course_id

course_id

course_id
CS-101
FIN-201
MU-199
HIS-351
CS-319
CS-319
CS-315

course_id
CS-101
CS-347
PHY-101
FIN-201
MU-199
HIS-351
CS-319
CS-315

Discussion:

SQL's set operations provide powerful tools for data modification and merging. Combining different values from two tables is successfully demonstrated by the above query. This technique can be very useful in a variety of analytical settings, such as identifying overlapping data, combining information from many sources, and locating unique records.

Experiment No:13

Experiment Name: Write queries using set membership, set comparison and test for empty relationship.

Introduction:

Developing SQL queries that apply set comparison, set membership, and empty relationship testing is the aim of this assignment. By examining these procedures, participants will have a better understanding of how SQL handles set-based operations, which will improve their ability to manage and analyze data in relational databases.

Program Code:

```
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 name from instructor where name not in ("Mozart", "Einstein");

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 dept_name
from instructor group by dept_name
having avg (salary) >= all (select avg (salary) from instructor group by dept_name);

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);

select distinct S.ID, S.name from student as S
where not exists ((select course_id from course where dept_name = "Biology")
```

except (select T.course_id from takes as T where S.ID = T.ID))

Output:

course_id
CS-101

course_id
CS-347
PHY-101

course_id
CS-101

name
Srinivasan
Wu
El Said
Gold
Katz
Califieri
Singh
Crick
Brandt
Kim

name
Wu
Einstein
Gold
Katz
Singh
Brandt
Kim

name
Wu
Einstein
Gold
Katz
Singh
Brandt
Kim

name
Wu
Einstein
Gold
Katz
Singh
Brandt
Kim

ID	name

Discussion:

The versatility of these database management techniques is demonstrated by the efficiency of set membership, set comparison, and testing for empty relationships in SQL, which are made possible by XAMPP software.

Experiment No:14

Experiment Name: Write queries on multiple relation and the using of 'natural join' keyword.

Introduction:

In relational databases, the NATURAL JOIN method automatically locates and matches columns with similar names and compatible data types, enabling the combination of two or more tables according to their shared attributes. This study demonstrates how to use NATURAL JOIN on many relations to get insightful information from a university database.

Program Code:

```
SELECT name AS instructor_name, title AS course_title, semester
FROM instructor NATURAL JOIN teaches NATURAL JOIN course;

SELECT room_number, capacity, course_id, sec_id
FROM classroom NATURAL JOIN section;

SELECT room_number, capacity, course_id, sec_id
FROM classroom NATURAL JOIN section;
```

Output:

student_name	advisor_id
Zhang	45565
Zhang	10101
Zhang	76543
Zhang	22222
Zhang	22222
Zhang	45565
Zhang	98345
Zhang	98345
Zhang	76766
Shankar	45565
Shankar	10101
Shankar	76543
Shankar	22222
Shankar	22222

instructor_name	course_title	semester
Srinivasan	Intro. to Computer Science	Fall
Srinivasan	Robotics	Spring
Srinivasan	Databse System Concepts	Fall
Wu	Investment Banking	Spring
Mozart	Music Video Production	Spring
Einstein	Physical Principles	Fall
El Said	World History	Spring
Katz	Intro. to Computer Science	Spring
Katz	Image Processing	Spring
Crick	Intro. to Biology	Summer
Crick	Genetics	Summer
Brandt	Image Processing	Spring
Kin	Intro. to Digital Systems	Spring

room_number	capacity	course_id	sec_id
514	10	BIO-101	1
514	10	BIO-301	1
101	500	CS-101	1
101	500	CS-101	1
3128	70	CS-190	1
3128	70	CS-190	2
120	50	CS-315	1
100	30	CS-319	1
3128	70	CS-319	2
3128	70	CS-347	1
3128	70	EE-181	1
101	500	FIN-201	1
514	10	HIS-351	1
101	500	MU-199	1
100	30	PHY-101	1

Discussion:

The NATURAL JOIN keyword facilitates linked table merger by automatically selecting columns with matching names. In this study, NATURAL JOIN was used to determine the relationships between classrooms and sections, professors and the courses they teach, and students and advisers. The database structure must be carefully constructed to ensure that no unexpected column matches occur, even if it removes the need to explicitly express join requirements. Because NATURAL JOIN can yield unexpected results when tables share numerous properties, it is important to confirm the use case.

Experiment No:15

Experiment Name: Write queries using different types of join.

Introduction:

By exploring different join types, including INNER, LEFT, RIGHT, and FULL joins, the objective of this exercise is to become proficient with SQL queries. By knowing these join types, participants may effectively access and integrate data from multiple tables and gain a comprehensive grasp of relational database queries.

Program Code:

```
Select * from student join takes on student.ID= takes.ID;
Select * from student, takes where student.ID = takes.ID;
Select * from student natural left outer join takes;
Select ID from student natural left outer join takes where course_id is null;
Select * from takes natural right outer join student;
```

Output:

ID
70557

ID	name	dept_nam	tot_cred	ID	course_id	sec_id	semester	year	grade
128	Zhang	Comp. Sci.	102	128	CS-101	1	Fall	2017	A
128	Zhang	Comp. Sci.	102	128	CS-347	1	Fall	2017	A
12345	Shankar	Comp. Sci.	32	12345	CS-101	1	Fall	2017	C
12345	Shankar	Comp. Sci.	32	12345	CS-190	2	Spring	2017	A
12345	Shankar	Comp. Sci.	32	12345	CS-315	1	Spring	2018	A
12345	Shankar	Comp. Sci.	32	12345	CS-347	1	Fall	2017	A
19991	Brandt	History	80	19991	HIS-351	1	Spring	2018	B
23121	Chavez	Finance	110	23121	FIN-201	1	Spring	2018	C+
44553	Peltier	Physics	56	44553	PHY-101	1	Fall	2017	B
45678	Levy	Physics	46	45678	CS-101	1	Fall	2017	F
45678	Levy	Physics	46	45678	CS-101	1	Spring	2018	B+
45678	Levy	Physics	46	45678	CS-319	1	Spring	2018	B
54321	Williams	Comp. Sci.	54	54321	CS-101	1	Fall	2017	A
54321	Williams	Comp. Sci.	54	54321	CS-190	2	Spring	2017	B+
55739	Sanchez	Music	38	55739	MU-199	1	Spring	2018	A
76543	Brown	Comp. Sci.	58	76543	CS-101	1	Fall	2017	A
76543	Brown	Comp. Sci.	58	76543	CS-319	2	Spring	2018	A
76653	Aoi	Elec. Eng.	60	76653	EE-181	1	Spring	2017	C
98765	Bourikas	Elec. Eng.	98	98765	CS-101	1	Fall	2017	C
98765	Bourikas	Elec. Eng.	98	98765	CS-315	1	Spring	2018	B
98988	Tanaka	Biology	120	98988	BIO-101	1	Summer	2017	A
98988	Tanaka	Biology	120	98988	BIO-301	1	Summer	2018	NULL

ID	name	dept_nam	tot_cred	ID	course_id	sec_id	semester	year	grade
128	Zhang	Comp. Sci.	102	128	CS-101	1	Fall	2017	A
128	Zhang	Comp. Sci.	102	128	CS-347	1	Fall	2017	A
12345	Shankar	Comp. Sci.	32	12345	CS-101	1	Fall	2017	C
12345	Shankar	Comp. Sci.	32	12345	CS-190	2	Spring	2017	A
12345	Shankar	Comp. Sci.	32	12345	CS-315	1	Spring	2018	A
12345	Shankar	Comp. Sci.	32	12345	CS-347	1	Fall	2017	A
19991	Brandt	History	80	19991	HIS-351	1	Spring	2018	B
23121	Chavez	Finance	110	23121	FIN-201	1	Spring	2018	C+
44553	Peltier	Physics	56	44553	PHY-101	1	Fall	2017	B
45678	Levy	Physics	46	45678	CS-101	1	Fall	2017	F
45678	Levy	Physics	46	45678	CS-101	1	Spring	2018	B+
45678	Levy	Physics	46	45678	CS-319	1	Spring	2018	B
54321	Williams	Comp. Sci.	54	54321	CS-101	1	Fall	2017	A
54321	Williams	Comp. Sci.	54	54321	CS-190	2	Spring	2017	B+
55739	Sanchez	Music	38	55739	MU-199	1	Spring	2018	A
76543	Brown	Comp. Sci.	58	76543	CS-101	1	Fall	2017	A
76543	Brown	Comp. Sci.	58	76543	CS-319	2	Spring	2018	A
76653	Aoi	Elec. Eng.	60	76653	EE-181	1	Spring	2017	C
98765	Bourikas	Elec. Eng.	98	98765	CS-101	1	Fall	2017	C
98765	Bourikas	Elec. Eng.	98	98765	CS-315	1	Spring	2018	B
98988	Tanaka	Biology	120	98988	BIO-101	1	Summer	2017	A
98988	Tanaka	Biology	120	98988	BIO-301	1	Summer	2018	NULL

ID	name	dept_nam	tot_cred	course_id	sec_id	semester	year	grade
128	Zhang	Comp. Sci.	102	CS-101	1	Fall	2017	A
128	Zhang	Comp. Sci.	102	CS-347	1	Fall	2017	A
12345	Shankar	Comp. Sci.	32	CS-101	1	Fall	2017	C
12345	Shankar	Comp. Sci.	32	CS-190	2	Spring	2017	A
12345	Shankar	Comp. Sci.	32	CS-315	1	Spring	2018	A
12345	Shankar	Comp. Sci.	32	CS-347	1	Fall	2017	A
19991	Brandt	History	80	HIS-351	1	Spring	2018	B
23121	Chavez	Finance	110	FIN-201	1	Spring	2018	C+
44553	Peltier	Physics	56	PHY-101	1	Fall	2017	B
45678	Levy	Physics	46	CS-101	1	Fall	2017	F
45678	Levy	Physics	46	CS-101	1	Spring	2018	B+
45678	Levy	Physics	46	CS-319	1	Spring	2018	B
54321	Williams	Comp. Sci.	54	CS-101	1	Fall	2017	A
54321	Williams	Comp. Sci.	54	CS-190	2	Spring	2017	B+
55739	Sanchez	Music	38	MU-199	1	Spring	2018	A
70557	Snow	Physics	0	NULL	NULL	NULL	NULL	NULL
76543	Brown	Comp. Sci.	58	CS-101	1	Fall	2017	A
76543	Brown	Comp. Sci.	58	CS-319	2	Spring	2018	A
76653	Aoi	Elec. Eng.	60	EE-181	1	Spring	2017	C
98765	Bourikas	Elec. Eng.	98	CS-101	1	Fall	2017	C
98765	Bourikas	Elec. Eng.	98	CS-315	1	Spring	2018	B
98988	Tanaka	Biology	120	BIO-101	1	Summer	2017	A
98988	Tanaka	Biology	120	BIO-301	1	Summer	2018	NULL

ID	name	dept_nam	tot_cred	course_id	sec_id	semester	year	grade
128	Zhang	Comp. Sci.	102	CS-101	1	Fall	2017	A
128	Zhang	Comp. Sci.	102	CS-347	1	Fall	2017	A
12345	Shankar	Comp. Sci.	32	CS-101	1	Fall	2017	C
12345	Shankar	Comp. Sci.	32	CS-190	2	Spring	2017	A
12345	Shankar	Comp. Sci.	32	CS-315	1	Spring	2018	A
12345	Shankar	Comp. Sci.	32	CS-347	1	Fall	2017	A
19991	Brandt	History	80	HIS-351	1	Spring	2018	B
23121	Chavez	Finance	110	FIN-201	1	Spring	2018	C+
44553	Peltier	Physics	56	PHY-101	1	Fall	2017	B
45678	Levy	Physics	46	CS-101	1	Fall	2017	F
45678	Levy	Physics	46	CS-101	1	Spring	2018	B+
45678	Levy	Physics	46	CS-319	1	Spring	2018	B
54321	Williams	Comp. Sci.	54	CS-101	1	Fall	2017	A
54321	Williams	Comp. Sci.	54	CS-190	2	Spring	2017	B+
55739	Sanchez	Music	38	MU-199	1	Spring	2018	A
70557	Snow	Physics	0	NULL	NULL	NULL	NULL	NULL
76543	Brown	Comp. Sci.	58	CS-101	1	Fall	2017	A
76543	Brown	Comp. Sci.	58	CS-319	2	Spring	2018	A
76653	Aoi	Elec. Eng.	60	EE-181	1	Spring	2017	C
98765	Bourikas	Elec. Eng.	98	CS-101	1	Fall	2017	C
98765	Bourikas	Elec. Eng.	98	CS-315	1	Spring	2018	B
98988	Tanaka	Biology	120	BIO-101	1	Summer	2017	A
98988	Tanaka	Biology	120	BIO-301	1	Summer	2018	NULL

Discussion:

The fact that several SQL joins were successfully applied in this experiment highlights how crucial they are for data retrieval. By mastering INNER, LEFT, RIGHT, and FULL joins, participants get a thorough grasp of combining data from linked tables. This ability increases query flexibility and accuracy, both of which are critical for database administration.

Experiment No:16

Experiment Name:
Write SQL queries to create and manipulate views for displaying student details with their respective course titles and grades.

Introduction:
The purpose of database indexes is to speed up the retrieval of data. In this report, we will create an index on the room_number column of the classroom database in order to optimize search queries based on classroom room numbers. If the index is no longer needed, we will also demonstrate how to remove it to guarantee efficient resource management.

Program Code:
```
CREATE INDEX idx_room_number ON classroom (room_number);

SELECT * FROM classroom WHERE room_number = '101';

DROP INDEX idx_room_number ON classroom;

SELECT * FROM classroom WHERE room_number = '101';

DROP INDEX idx_room_number ON classroom;
```

Output:

				building	room_number	capacity
☐	Edit	Copy	Delete	Packard	101	500
☐	Edit	Copy	Delete	Painter	514	10
☐	Edit	Copy	Delete	Taylor	3128	70
☐	Edit	Copy	Delete	Watson	100	30
☐	Edit	Copy	Delete	Watson	120	50

Disuccsion:
Creating an index on the room_number column improves the performance of queries that use room numbers to find specific classrooms. Indexes are particularly useful in large datasets, where searches would typically be slower. However, there are drawbacks to indexes, such as the requirement for more storage space and a slight decrease in speed when updates, deletions, and insertions are made. Dropping an index that is no longer needed can help save storage space and improve speed by eliminating unnecessary overhead.