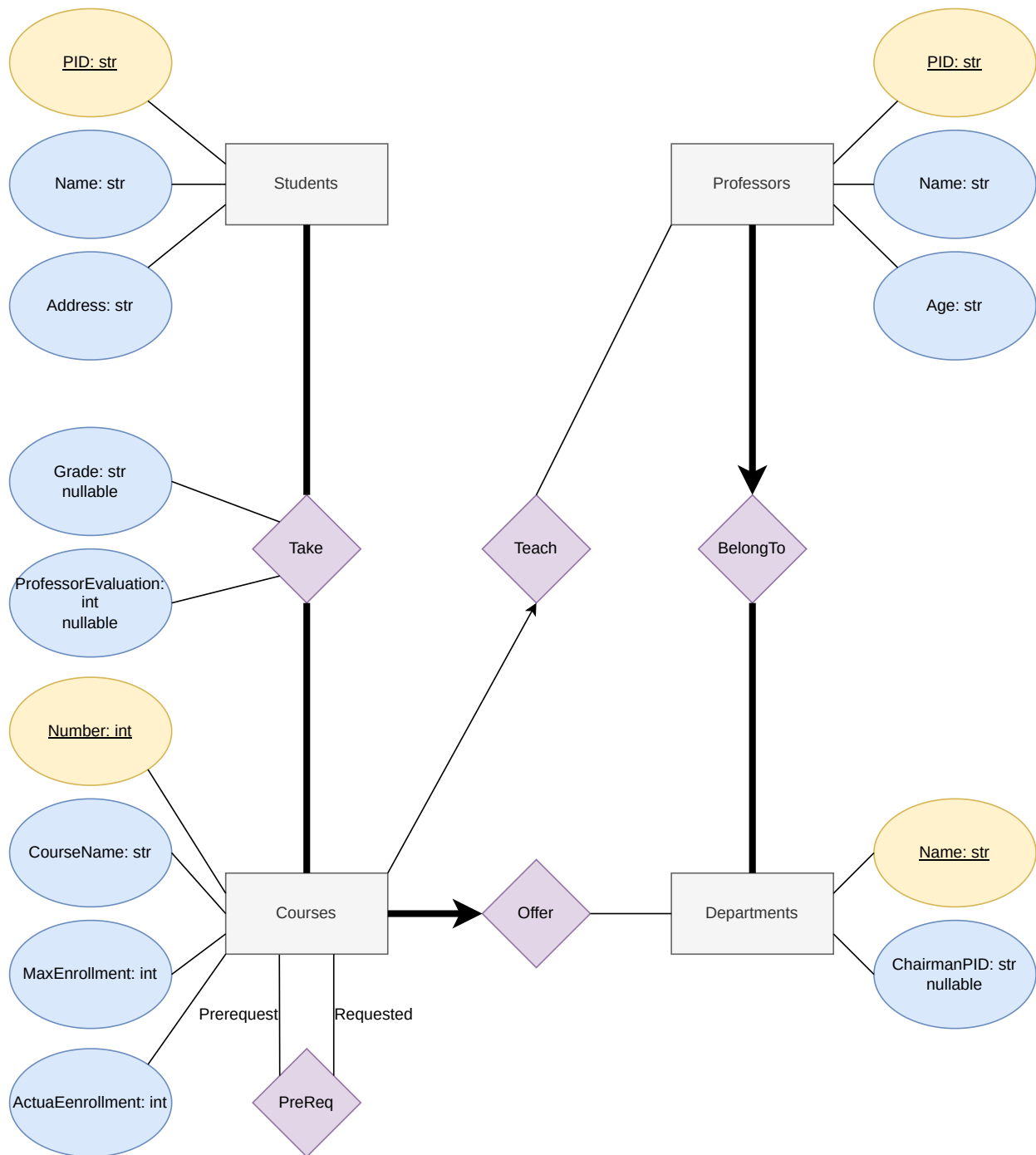


Author Information

- Name: 鍾博丞
- Student ID: 408410120
- E-mail: my072814638@csie.io

Part A

E-R Diagram



DDL

```

1 CREATE DATABASE IF NOT EXISTS `courses`;
2 USE `courses`;
3
4 -- Entities
5 CREATE TABLE IF NOT EXISTS `Students` (
6     `PID` VARCHAR(100) NOT NULL PRIMARY KEY,
7     `Name` TEXT NOT NULL,
8     `Address` TEXT NOT NULL
9 );
10

```

```

11 CREATE TABLE IF NOT EXISTS `Professors` (
12     `PID` VARCHAR(100) NOT NULL PRIMARY KEY,
13     `Name` TEXT NOT NULL,
14     `Age` TEXT NOT NULL,
15     `DepartmentName` VARCHAR(100) NOT NULL
16 );
17
18 CREATE TABLE IF NOT EXISTS `Departments` (
19     `Name` VARCHAR(100) NOT NULL PRIMARY KEY,
20     `ChairmanPID` VARCHAR(100) DEFAULT NULL COMMENT 'there are times when a
    department may not have a chairperson'
21 );
22
23 -- Circular References
24 ALTER TABLE `Professors`
25 ADD CONSTRAINT `FK_DepartmentName`
26 FOREIGN KEY (`DepartmentName`)
27     REFERENCES `Departments`(`Name`)
28     ON DELETE CASCADE
29     ON UPDATE CASCADE;
30
31 ALTER TABLE `Departments`
32 ADD CONSTRAINT `FK_ChairmanPID`
33 FOREIGN KEY (`ChairmanPID`)
34     REFERENCES `Professors`(`PID`)
35     ON DELETE CASCADE
36     ON UPDATE CASCADE;
37
38
39 CREATE TABLE IF NOT EXISTS `Courses` (
40     `Number` INT NOT NULL PRIMARY KEY,
41     `DeptName` VARCHAR(100) NOT NULL,
42     `CourseName` TEXT NOT NULL,
43     `MaxEnrollment` INT NOT NULL,
44     `ActualEnrollment` INT NOT NULL DEFAULT 0,
45     CONSTRAINT `check_enrollment`
46         CHECK (`ActualEnrollment` <= `MaxEnrollment`),
47     FOREIGN KEY (`DeptName`)
48         REFERENCES `Departments`(`Name`)
49         ON DELETE CASCADE
50         ON UPDATE CASCADE
51 ) COMMENT 'The actual enrollment must be at most the maximum enrollment.';
52
53 -- Relations
54 CREATE TABLE IF NOT EXISTS `Take` (
55     `StudentPID` VARCHAR(100) NOT NULL,
56     `Number` INT NOT NULL COMMENT 'Course Number',
57     `DeptName` VARCHAR(100) NOT NULL,
58     `Grade` TEXT DEFAULT NULL,
59     `ProfessorEvaluation` INT DEFAULT NULL,
60     PRIMARY KEY (`StudentPID`, `Number`, `DeptName`),
61     FOREIGN KEY (`StudentPID`)

```

```

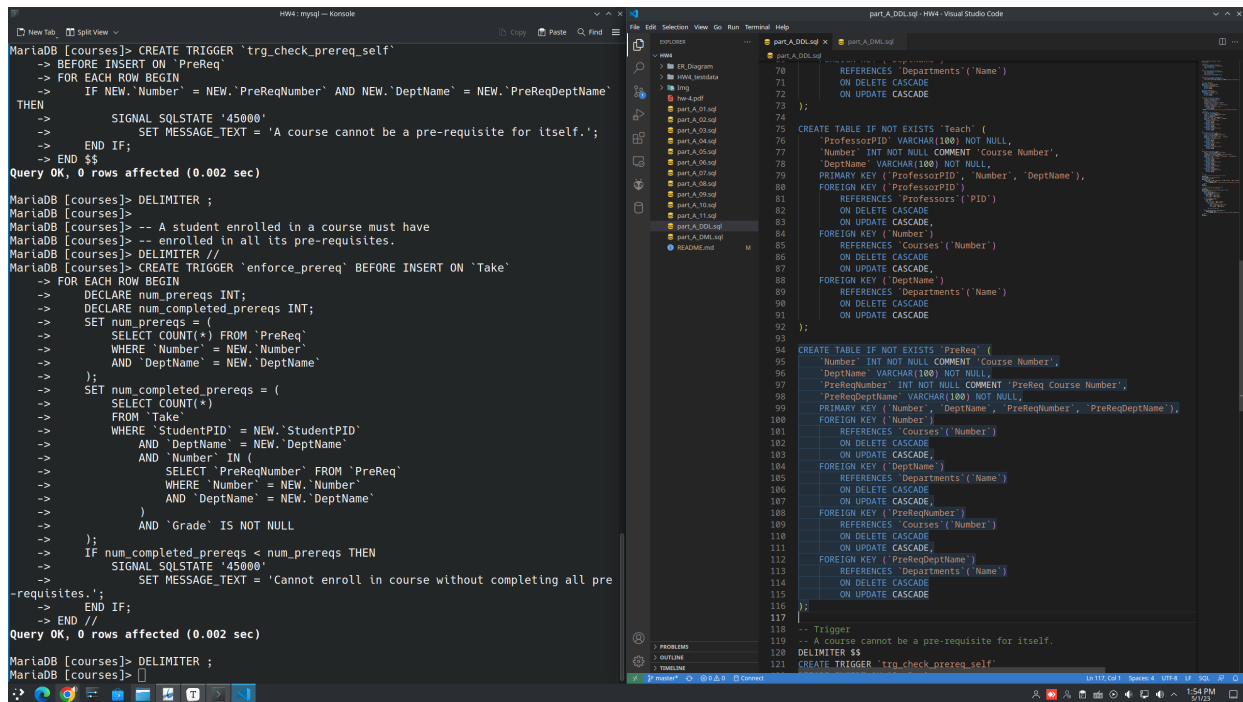
62         REFERENCES `Students`(`PID`)
63         ON DELETE CASCADE
64         ON UPDATE CASCADE,
65     FOREIGN KEY (`Number`)
66         REFERENCES `Courses`(`Number`)
67         ON DELETE CASCADE
68         ON UPDATE CASCADE,
69     FOREIGN KEY (`DeptName`)
70         REFERENCES `Departments`(`Name`)
71         ON DELETE CASCADE
72         ON UPDATE CASCADE
73 );
74
75 CREATE TABLE IF NOT EXISTS `Teach` (
76     `ProfessorPID` VARCHAR(100) NOT NULL,
77     `Number` INT NOT NULL COMMENT 'Course Number',
78     `DeptName` VARCHAR(100) NOT NULL,
79     PRIMARY KEY (`ProfessorPID`, `Number`, `DeptName`),
80     FOREIGN KEY (`ProfessorPID`)
81         REFERENCES `Professors`(`PID`)
82         ON DELETE CASCADE
83         ON UPDATE CASCADE,
84     FOREIGN KEY (`Number`)
85         REFERENCES `Courses`(`Number`)
86         ON DELETE CASCADE
87         ON UPDATE CASCADE,
88     FOREIGN KEY (`DeptName`)
89         REFERENCES `Departments`(`Name`)
90         ON DELETE CASCADE
91         ON UPDATE CASCADE
92 );
93
94 CREATE TABLE IF NOT EXISTS `PreReq` (
95     `Number` INT NOT NULL COMMENT 'Course Number',
96     `DeptName` VARCHAR(100) NOT NULL,
97     `PreReqNumber` INT NOT NULL COMMENT 'PreReq Course Number',
98     `PreReqDeptName` VARCHAR(100) NOT NULL,
99     PRIMARY KEY (`Number`, `DeptName`, `PreReqNumber`, `PreReqDeptName`),
100     FOREIGN KEY (`Number`)
101         REFERENCES `Courses`(`Number`)
102         ON DELETE CASCADE
103         ON UPDATE CASCADE,
104     FOREIGN KEY (`DeptName`)
105         REFERENCES `Departments`(`Name`)
106         ON DELETE CASCADE
107         ON UPDATE CASCADE,
108     FOREIGN KEY (`PreReqNumber`)
109         REFERENCES `Courses`(`Number`)
110         ON DELETE CASCADE
111         ON UPDATE CASCADE,
112     FOREIGN KEY (`PreReqDeptName`)
113         REFERENCES `Departments`(`Name`)

```

```

114         ON DELETE CASCADE
115         ON UPDATE CASCADE
116     );
117
118     -- Trigger
119     -- A course cannot be a pre-requisite for itself.
120     DELIMITER $$
121     CREATE TRIGGER `trg_check_prereq_self`
122     BEFORE INSERT ON `PreReq`
123     FOR EACH ROW BEGIN
124         IF NEW.`Number` = NEW.`PreReqNumber` AND NEW.`DeptName` =
125         NEW.`PreReqDeptName` THEN
126             SIGNAL SQLSTATE '45000'
127             SET MESSAGE_TEXT = 'A course cannot be a pre-requisite for
128             itself.';
129         END IF;
130     END $$
131     DELIMITER ;
132
133     -- A student enrolled in a course must have
134     -- enrolled in all its pre-requisites.
135     DELIMITER //
136     CREATE TRIGGER `enforce_prereq` BEFORE INSERT ON `Take`
137     FOR EACH ROW BEGIN
138         DECLARE num_prereqs INT;
139         DECLARE num_completed_prereqs INT;
140         SET num_prereqs = (
141             SELECT COUNT(*) FROM `PreReq`
142             WHERE `Number` = NEW.`Number`
143             AND `DeptName` = NEW.`DeptName`
144         );
145         SET num_completed_prereqs = (
146             SELECT COUNT(*)
147             FROM `Take`
148             WHERE `StudentPID` = NEW.`StudentPID`
149             AND `DeptName` = NEW.`DeptName`
150             AND `Number` IN (
151                 SELECT `PreReqNumber` FROM `PreReq`
152                 WHERE `Number` = NEW.`Number`
153                 AND `DeptName` = NEW.`DeptName`
154             )
155             AND `Grade` IS NOT NULL
156         );
157         IF num_completed_prereqs < num_prereqs THEN
158             SIGNAL SQLSTATE '45000'
159             SET MESSAGE_TEXT = 'Cannot enroll in course without completing all
160             pre-requisites.';
161         END IF;
162     END //
163     DELIMITER ;

```



DML

```

1  USE `courses`;
2
3  -- Delete all rows for every tables.
4  DELETE FROM `Students`;
5  DELETE FROM `Professors`;
6  DELETE FROM `Departments`;
7  DELETE FROM `Courses`;
8  DELETE FROM `Take`;
9  DELETE FROM `Teach`;
10 DELETE FROM `PreReq`;
11
12 INSERT INTO `Students`(`PID`, `Name`, `Address`) VALUES
13     ('Zadeh',      'Lofti',      'Seattle, WA'),
14     ('Patterson',  'David',      'Los Angeles, CA'),
15     ('Smith',     'Alan',       'San Francisco, CA'),
16     ('Feiner',    'Steven',    'Boston, MA'),
17     ('Kuck',      'David',      'Bloomington, IN'),
18     ('Kender',    'John',       'Los Angeles, CA'),
19     ('Huang',     'Thomas',    'Atlanta, GA'),
20     ('Fischer',   'Michael',   'Madison, WI'),
21     ('Appel',     'Andrew',   'Miami, FL'),
22     ('Dobkin',    'David',    'Salt Lake City, UT'),
23     ('Li',        'Kai',        'Las Vegas, NV'),
24     ('Peterson',  'Larry',    'Chicago, IL');
25
26 INSERT INTO `Departments`(`Name`, `ChairmanPID`) VALUES
27     ('CS',      NULL),
28     ('EE',      NULL),

```

```

29      ('ME',    NULL),
30      ('BIO',   NULL),
31      ('PHY',   NULL),
32      ('MATH',  NULL);
33
34  INSERT INTO `Professors`(`PID`, `Name`, `Age`, `DepartmentName`) VALUES
35      ('Widom',   'Jennifer', 'old',      'BIO'),
36      ('Canny',   'John',     'very old', 'EE'),
37      ('Ullman',  'Jeff',     'still alive', 'CS'),      -- Chairman
38      ('Reiss',   'Steve',    'very old', 'PHY'),      -- Chairman
39      ('Karp',    'Richard',  'still alive', 'MATH'),
40      ('Lam',     'Monica',    'old',      'ME'),      -- Chairman
41      ('Chien',   'Andrew',   'old',      'PHY'),
42      ('Wegner',  'Peter',    'still alive', 'MATH'),    -- Chairman
43      ('Hart',    'John',     'very old', 'BIO'),
44      ('Katz',    'Randy',    'very old', 'CS'),
45      ('Knuth',   'Don',      'still alive', 'EE'),      -- Chairman
46      ('Barsky',  'Brian',    'old',      'EE');
47
48  UPDATE `Departments` SET `ChairmanPID` = 'Ullman' WHERE `Name` = 'CS';
49  UPDATE `Departments` SET `ChairmanPID` = 'Knuth' WHERE `Name` = 'EE';
50  UPDATE `Departments` SET `ChairmanPID` = 'Lam' WHERE `Name` = 'ME';
51  UPDATE `Departments` SET `ChairmanPID` = 'Reiss' WHERE `Name` = 'PHY';
52  UPDATE `Departments` SET `ChairmanPID` = 'Wegner' WHERE `Name` = 'MATH';
53
54  INSERT INTO `Courses`(`Number`, `DeptName`, `CourseName`, `MaxEnrollment`,
55      `ActualEnrollment`) VALUES
56      (132, 'ME', 'Dynamic Systems', 120, 118),
57      (61, 'CS', 'Data Structure', 100, 90),
58      (1, 'MATH', 'Calculus', 150, 132),
59      (123, 'EE', 'Digital Signal Proc', 80, 72),
60      (111, 'PHY', 'Modern Physics', 40, 39),
61      (109, 'ME', 'Heat Transfer', 10, 8),
62      (54, 'MATH', 'Linear Algebra', 50, 50),
63      (162, 'CS', 'Operating Systems', 50, 32),
64      (137, 'PHY', 'Quantum Mech', 10, 3),
65      (145, 'BIO', 'Genomics', 5, 2),
66      (186, 'CS', 'Database Systems', 50, 48),
67      (224, 'EE', 'Digital Comm', 30, 22);
68
69  INSERT INTO `Teach`(`ProfessorPID`, `Number`, `DeptName`) VALUES
70      ('Knuth', 123, 'EE'),
71      ('Reiss', 54, 'MATH'),
72      ('Widom', 145, 'BIO'),
73      ('Ullman', 61, 'CS'),
74      ('Karp', 224, 'EE'),
75      ('Lam', 132, 'ME'),
76      ('Reiss', 111, 'PHY'),
77      ('Wegner', 1, 'MATH'),
78      ('Ullman', 186, 'CS'),
79      ('Reiss', 137, 'PHY'),

```

```

80      ('Chien', 109, 'ME'),
81      ('Barsky', 162, 'CS');
82
83  INSERT INTO `PreReq`(`Number`, `DeptName`, `PreReqNumber`, `PreReqDeptName`)
      VALUES
84      (111, 'PHY', 1, 'MATH'),
85      (137, 'PHY', 1, 'MATH'),
86      (123, 'EE', 54, 'MATH'),
87      (186, 'CS', 54, 'MATH'),
88      (224, 'EE', 54, 'MATH'),
89      (186, 'CS', 61, 'CS'),
90      (224, 'EE', 61, 'CS'),
91      (111, 'PHY', 132, 'ME'),
92      (145, 'BIO', 132, 'ME'),
93      (132, 'ME', 145, 'BIO'),
94      ( 54, 'MATH', 162, 'CS'),
95      (162, 'CS', 186, 'CS'),
96      (109, 'ME', 224, 'EE');
97
98
99  INSERT INTO `Take`(`StudentPID`, `Number`, `DeptName`, `Grade`,
      `ProfessorEvaluation`) VALUES
100      ('Appel', 111, 'PHY', 'B', 2),
101      ('Patterson', 186, 'CS', 'B', 3),
102      ('Li', 137, 'PHY', 'A', 3),
103      ('Huang', 186, 'CS', 'A', 4),
104      ('Smith', 109, 'ME', 'A', 3),
105      ('Appel', 1, 'MATH', 'C', 2),
106      ('Huang', 123, 'EE', 'A', 4),
107      ('Fischer', 145, 'BIO', 'A', 2),
108      ('Zadeh', 61, 'CS', 'A', 1),
109      ('Dobkin', 123, 'EE', 'B', 4),
110      ('Huang', 111, 'PHY', 'B', 3),
111      ('Li', 162, 'CS', 'A', 3),
112      ('Kender', 54, 'MATH', 'B', 4);

```



```
--> ('Ullman', 186, 'CS'),
--> ('Reiss', 137, 'PHY'),
--> ('Chien', 109, 'ME'),
--> ('Barsky', 162, 'CS');
Query OK, 12 rows affected (0.000 sec)
Records: 12 Duplicates: 0 Warnings: 0

MariaDB [courses]>
MariaDB [courses]> INSERT INTO `PreReq`(`Number`, `DeptName`, `PreReqNumber`, `PreReqDept
Name`) VALUES
--> (111, 'PHY', 1, 'MATH'),
--> (137, 'PHY', 1, 'MATH'),
--> (123, 'EE', 54, 'MATH'),
--> (186, 'CS', 54, 'MATH'),
--> (224, 'EE', 54, 'MATH'),
--> (186, 'CS', 61, 'CS'),
--> (224, 'EE', 61, 'CS'),
--> (111, 'PHY', 132, 'ME'),
--> (145, 'PHY', 132, 'ME'),
--> (132, 'ME', 145, 'BIO'),
--> (54, 'MATH', 162, 'CS'),
--> (162, 'CS', 186, 'CS'),
--> (189, 'ME', 224, 'EE');
Query OK, 13 rows affected (0.001 sec)
Records: 13 Duplicates: 0 Warnings: 0

MariaDB [courses]>
MariaDB [courses]>
MariaDB [courses]> INSERT INTO `Take`(`StudentPID`, `Number`, `DeptName`, `Grade`, `Profe
ssorEvaluation`) VALUES
--> ('Appel', 111, 'PHY', 'B', 2),
--> ('Patterson', 186, 'CS', 'B', 3),
--> ('Li', 137, 'PHY', 'A', 3),
--> ('Huang', 186, 'CS', 'A', 4),
--> ('Smith', 109, 'ME', 'A', 3),
--> ('Appel', 1, 'MATH', 'C', 2),
--> ('Huang', 123, 'EE', 'A', 4),
--> ('Fischer', 145, 'BIO', 'A', 2),
--> ('Zadeh', 61, 'CS', 'A', 1),
--> ('Dobkin', 123, 'EE', 'B', 4),
--> ('Huang', 111, 'PHY', 'B', 3),
--> ('Li', 162, 'CS', 'A', 3),
--> ('Kender', 54, 'MATH', 'B', 4);
Query OK, 13 rows affected (0.000 sec)
Records: 13 Duplicates: 0 Warnings: 0
```

Usage

Circular Dependency

[Circular Dependency in FK](#)

The attribute "DepartmentName" in table "Professors" references the attribute "Name" in table "Departments".

The attribute "ChairmanPID" in table "Departments" references the attribute "PID" in table "Professors".

This will cause circular dependency in foreign key constraint.

To solve it, first insert new departments with NULL chairman, then insert professors. Finally, update the chairmans of departments.

Prerequisite Trigger

Since there are cycle dependencies in prerequest, you have to disable the trigger before you want to insert a new row into table "Take".

1

What are the PIDs of the students whose name is "David"?

```
1 SELECT `PID`  
2 FROM `Students`  
3 WHERE `name` = 'David';
```

Ans: Dobkin, Kuck, Patterson.

2

Which pairs of students live at the same address? It is enough to return the names of such students pairs.

```
1 SELECT s1.`PID`, s1.`Name`, s2.`PID`, s2.`Name`  
2 FROM `Students` s1  
3 INNER JOIN `Students` s2 ON s1.`Address` = s2.`Address`  
4 WHERE s1.`PID` < s2.`PID`;
```

Ans: Kender John and Patterson David.

3

Which department have course that have pre-requisites in other departments?

```
1 SELECT DISTINCT d.`Name` AS 'Department Name'  
2 FROM `Departments` d  
3 INNER JOIN `Courses` c ON d.`Name` = c.`DeptName`  
4 INNER JOIN `PreReq` p ON c.`Number` = p.`Number` AND c.`DeptName` = p.`DeptName`  
5 INNER JOIN `Departments` dp ON p.`PreReqDeptName` = dp.`Name`  
6 WHERE dp.`Name` <> d.`Name`;
```

Ans: BIO, CS, EE, MATH, ME, PHY.

4

[Stack Overflow Recursive Query](#), [MySQL Recursive CTE](#), [MySQL Blog Archive](#)

Compute the set of all courses that are their own pre-requisites? (have cycles)

```
1 WITH RECURSIVE `descendants` AS (  
2     SELECT p.`DeptName`, p.`Number`, CONCAT(CAST(p.`DeptName` AS CHAR(1000000)),  
3     ' ', CAST(p.`Number` AS CHAR(500))) AS `Path`, 0 AS `is_cycle`  
4     FROM `PreReq` p
```

```

4      WHERE p.`PreReqNumber` = 54 AND p.`PreReqDeptName` = 'MATH'
5      UNION ALL
6      SELECT p.`DeptName`, p.`Number`, CONCAT(CAST(p.`DeptName` AS CHAR(1000000)),
7      ' ', CAST(p.`Number` AS CHAR(500))) AS `Path`, 0 AS `is_cycle`
8      FROM `PreReq` p
9      WHERE p.`PreReqNumber` = 61 AND p.`PreReqDeptName` = 'CS'
10     UNION ALL
11     SELECT p.`DeptName`, p.`Number`, CONCAT(CAST(p.`DeptName` AS CHAR(1000000)),
12     ' ', CAST(p.`Number` AS CHAR(500))) AS `Path`, 0 AS `is_cycle`
13     FROM `PreReq` p
14     WHERE p.`PreReqNumber` = 1 AND p.`PreReqDeptName` = 'MATH'
15     UNION ALL
16     SELECT p.`DeptName`, p.`Number`, CONCAT(CAST(p.`DeptName` AS CHAR(1000000)),
17     ' ', CAST(p.`Number` AS CHAR(500))) AS `Path`, 0 AS `is_cycle`
18     FROM `PreReq` p
19     WHERE p.`PreReqNumber` = 132 AND p.`PreReqDeptName` = 'ME'
20     UNION ALL
21     SELECT p2.`DeptName`, p2.`Number`, CONCAT(d.`Path`, ' ', p2.`DeptName`, ' ',
22     p2.`Number`), FIND_IN_SET(CONCAT(p2.`DeptName`, ' ', p2.`Number`), d.`Path`)
23     != 0
24     FROM `PreReq` p2, `descendants` d
25     WHERE p2.`PreReqNumber` = d.`Number` AND p2.`PreReqDeptName` = d.`DeptName`
26     AND `is_cycle` = 0
27 )
28 SELECT * FROM `descendants`;

```

Ans:

DeptName	Number	Path	is_cycle
EE	123	EE 123	0
CS	186	CS 186	0
EE	224	EE 224	0
CS	186	CS 186	0
EE	224	EE 224	0
PHY	111	PHY 111	0
PHY	137	PHY 137	0
PHY	111	PHY 111	0
BIO	145	BIO 145	0
CS	162	CS 186, CS 162	0
ME	109	EE 224, ME 109	0
CS	162	CS 186, CS 162	0
ME	109	EE 224, ME 109	0
ME	132	BIO 145, ME 132	0
MATH	54	CS 186, CS 162, MATH 54	0
MATH	54	CS 186, CS 162, MATH 54	0
PHY	111	BIO 145, ME 132, PHY 111	0
BIO	145	BIO 145, ME 132, BIO 145	1
EE	123	CS 186, CS 162, MATH 54, EE 123	0
CS	186	CS 186, CS 162, MATH 54, CS 186	1

24		EE		224		CS 186, CS 162, MATH 54, EE 224		0		
25		EE		123		CS 186, CS 162, MATH 54, EE 123		0		
26		CS		186		CS 186, CS 162, MATH 54, CS 186		1		
27		EE		224		CS 186, CS 162, MATH 54, EE 224		0		
28		ME		109		CS 186, CS 162, MATH 54, EE 224, ME 109		0		
29		ME		109		CS 186, CS 162, MATH 54, EE 224, ME 109		0		
30		+-----+-----+-----+-----+-----+								

There are 2 cycle dependencies.

1. MATH 54, CS 186, CS 162, MATH 54
2. ME 132, BIO 145, ME 132

5

What are the names and address of the students who are taking "CS186"?

```

1 SELECT s.`PID`, s.`Name`, s.`Address`
2 FROM `Students` s
3 INNER JOIN `Take` t ON s.`PID` = t.`StudentPID`
4 WHERE t.`Number` = 186 AND t.`DeptName` = 'CS';

```

Ans:

1		+-----+-----+-----+		
2		PID	Name	Address
3		+-----+-----+-----+		
4		Huang	Thomas	Atlanta, GA
5		Patterson	David	Los Angeles, CA
6		+-----+-----+-----+		

6

What are the courses that the head of the CS department is teaching?

```

1 SELECT c.`CourseName`
2 FROM `Teach` t
3 INNER JOIN `Courses` c ON t.`Number` = c.`Number` AND t.`DeptName` = c.`DeptName`
4 INNER JOIN `Departments` d ON t.`DeptName` = d.`Name`
5 LEFT JOIN `Professors` p ON d.`ChairmanPID` = p.`PID`
6 WHERE t.`DeptName` = 'CS' AND p.`PID` IS NOT NULL AND d.`ChairmanPID` = p.`PID`;

```

Ans: Operating Systems, Data Structure, and Database Systems.

7

Is there any department head who teaches a course in another department?

```
1 SELECT DISTINCT p.`PID`, p.`Name`
2 FROM `Professors` p
3 INNER JOIN `Departments` d ON p.`PID` = d.`ChairmanPID`
4 INNER JOIN `Teach` t ON p.`PID` = t.`ProfessorPID`
5 WHERE p.`DepartmentName` <> t.`DeptName`;
```

Ans: Yes. Reiss Steve is the chairman for department of physics, and he teaches "MATH 54".

8

Are there any students who are taking at least two courses taught by department heads?

```
1 SELECT `Result`.`PID`, `Result`.`Name`
2 FROM (
3     SELECT s.`PID`, s.`Name`, COUNT(*) AS `Amount`
4     FROM `Students` s
5     INNER JOIN `Take` t ON s.`PID` = t.`StudentPID`
6     INNER JOIN `Teach` te ON t.`Number` = te.`Number` AND t.`DeptName` =
7     te.`DeptName`
8     INNER JOIN `Departments` d ON te.`ProfessorPID` = d.`ChairmanPID`
9     GROUP BY s.`PID`
10 ) AS `Result`
11 WHERE `Result`.`Amount` >= 2;
```

Ans: Yes. There are Appel Andrew and Huang Thomas.

9

Is there any professor whose age is "still alive" and who receives an average evaluation above 2.5?

```
1 SELECT p.`PID`, p.`Name`, AVG(ta.`ProfessorEvaluation`) AS `Average_Evaluation`
2 FROM `Take` ta
3 INNER JOIN `Teach` te ON ta.`Number` = te.`Number` AND ta.`DeptName` =
4 te.`DeptName`
5 INNER JOIN `Professors` p ON te.`ProfessorPID` = p.`PID`
6 WHERE p.`Age` = 'still alive'
7 GROUP BY p.`PID`;
```

Ans: Yes.

```

1  +-----+-----+-----+
2  | PID    | Name  | Average_Evaluation |
3  +-----+-----+-----+
4  | Knuth  | Don   | 4.0000 |
5  | Ullman | Jeff  | 2.6667 |
6  | Wegner | Peter | 2.0000 |
7  +-----+-----+-----+

```

10

Is there any “straight A” student?

We define straight-A student to be the student who gets all A in his/her courses.

```

1  SELECT DISTINCT s.`PID`, s.`Name`
2  FROM `Students` s
3  INNER JOIN `Take` t ON s.`PID` = t.`StudentPID`
4  WHERE EXISTS (
5      SELECT *
6      FROM `Take` t1
7      WHERE t.`StudentPID` = t1.`StudentPID` AND NOT EXISTS (
8          SELECT *
9          FROM `Take` t2
10         WHERE t2.`StudentPID` = s.`PID` AND t2.`Grade` <> 'A'
11     )
12 );

```

Ans: Yes. There are Fischer Michael, Li Kai, Smith Alan, Zadeh Lofti

11

Are there any students who are taking courses and receiving more grade A than grade B?

```

1  SELECT s.`PID`, s.`Name`, `ResultA`.`Amount_of_A`, `ResultB`.`Amount_of_B`
2  FROM `Students` s
3  INNER JOIN (
4      SELECT t.`StudentPID`, COUNT(*) AS `Amount_of_A`
5      FROM `Take` t
6      WHERE t.`Grade` = 'A'
7      GROUP BY t.`StudentPID`
8  ) AS `ResultA` ON s.`PID` = `ResultA`.`StudentPID`
9  LEFT JOIN (
10     SELECT t.`StudentPID`, COUNT(*) AS `Amount_of_B`
11     FROM `Take` t
12     WHERE t.`Grade` = 'B'
13     GROUP BY t.`StudentPID`
14 ) AS `ResultB` ON s.`PID` = `ResultB`.`StudentPID`

```

```

15 WHERE `ResultA`.`Amount_of_A` > `ResultB`.`Amount_of_B`
16 OR (`ResultA`.`Amount_of_A` IS NOT NULL AND `ResultB`.`Amount_of_B` IS NULL);

```

Ans: Yes.

PID	Name	Amount_of_A	Amount_of_B
Fischer	Michael	1	NULL
Huang	Thomas	2	1
Li	Kai	2	NULL
Smith	Alan	1	NULL
Zadeh	Lofti	1	NULL

NULL means 0 here.

Part B

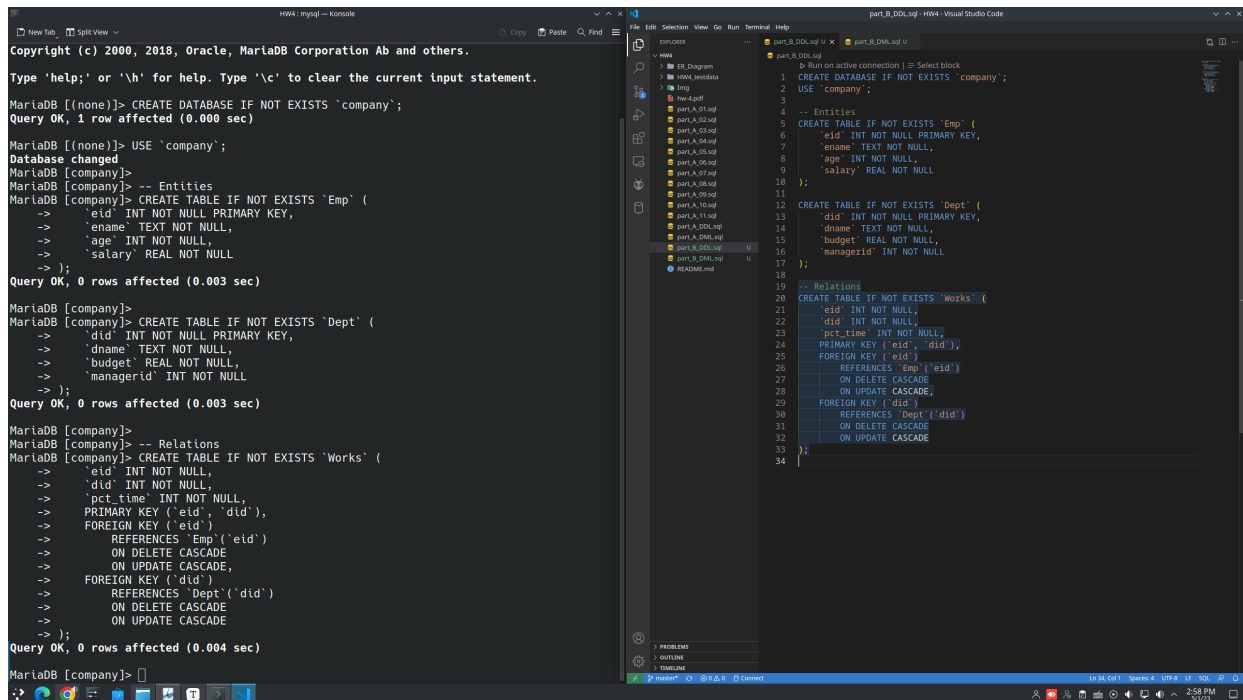
DDL

```

1  -- Entities
2  CREATE TABLE IF NOT EXISTS `Emp` (
3      `eid` INT NOT NULL PRIMARY KEY,
4      `ename` TEXT NOT NULL,
5      `age` INT NOT NULL,
6      `salary` REAL NOT NULL
7  );
8
9  CREATE TABLE IF NOT EXISTS `Dept` (
10     `did` INT NOT NULL PRIMARY KEY,
11     `dname` TEXT NOT NULL,
12     `budget` REAL NOT NULL,
13     `managerid` INT NOT NULL
14 );
15
16 -- Relations
17 CREATE TABLE IF NOT EXISTS `Works` (
18     `eid` INT NOT NULL,
19     `did` INT NOT NULL,
20     `pct_time` INT NOT NULL,
21     PRIMARY KEY (`eid`, `did`),
22     FOREIGN KEY (`eid`)
23         REFERENCES `Emp`(`eid`)
24         ON DELETE CASCADE
25         ON UPDATE CASCADE,
26     FOREIGN KEY (`did`)
27         REFERENCES `Dept`(`did`)
28         ON DELETE CASCADE

```

```
29 ON UPDATE CASCADE
30 );
```



The screenshot shows a Visual Studio Code editor with two main windows. The left window is a terminal running a MariaDB console session. The right window is a text editor showing a SQL script file named 'part_8_DDL.sql'.

MariaDB Console Output:

```
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> CREATE DATABASE IF NOT EXISTS 'company';
Query OK, 1 row affected (0.000 sec)

MariaDB [(none)]> USE 'company';
Database changed
MariaDB [company]>
MariaDB [company]> -- Entities
MariaDB [company]> CREATE TABLE IF NOT EXISTS 'Emp' (
->   'eid' INT NOT NULL PRIMARY KEY,
->   'ename' TEXT NOT NULL,
->   'age' INT NOT NULL,
->   'salary' REAL NOT NULL
-> );
Query OK, 0 rows affected (0.003 sec)

MariaDB [company]>
MariaDB [company]> CREATE TABLE IF NOT EXISTS 'Dept' (
->   'did' INT NOT NULL PRIMARY KEY,
->   'dname' TEXT NOT NULL,
->   'budget' REAL NOT NULL,
->   'managerid' INT NOT NULL
-> );
Query OK, 0 rows affected (0.003 sec)

MariaDB [company]>
MariaDB [company]> -- Relations
MariaDB [company]> CREATE TABLE IF NOT EXISTS 'Works' (
->   'eid' INT NOT NULL,
->   'did' INT NOT NULL,
->   'pct_time' INT NOT NULL,
->   PRIMARY KEY ('eid', 'did'),
->   FOREIGN KEY ('eid')
->     REFERENCES 'Emp' ('eid')
->     ON DELETE CASCADE
->     ON UPDATE CASCADE,
->   FOREIGN KEY ('did')
->     REFERENCES 'Dept' ('did')
->     ON DELETE CASCADE
->     ON UPDATE CASCADE
-> );
Query OK, 0 rows affected (0.004 sec)

MariaDB [company]>
```

SQL Script File (part_8_DDL.sql):

```
1 CREATE DATABASE IF NOT EXISTS 'company';
2 USE 'company';
3
4 -- Entities
5 CREATE TABLE IF NOT EXISTS 'Emp' (
6   'eid' INT NOT NULL PRIMARY KEY,
7   'ename' TEXT NOT NULL,
8   'age' INT NOT NULL,
9   'salary' REAL NOT NULL
10 );
11
12 CREATE TABLE IF NOT EXISTS 'Dept' (
13   'did' INT NOT NULL PRIMARY KEY,
14   'dname' TEXT NOT NULL,
15   'budget' REAL NOT NULL,
16   'managerid' INT NOT NULL
17 );
18
19 -- Relations
20 CREATE TABLE IF NOT EXISTS 'Works' (
21   'eid' INT NOT NULL,
22   'did' INT NOT NULL,
23   'pct_time' INT NOT NULL,
24   PRIMARY KEY ('eid', 'did'),
25   FOREIGN KEY ('eid')
26     REFERENCES 'Emp' ('eid')
27     ON DELETE CASCADE
28     ON UPDATE CASCADE,
29   FOREIGN KEY ('did')
30     REFERENCES 'Dept' ('did')
31     ON DELETE CASCADE
32     ON UPDATE CASCADE
33 );
34
```

DML

These are from sample data.

```
1 INSERT INTO `Emp`(`eid`, `ename`, `age`, `salary`) VALUES
2   ( 1, 'Alice',    28, 55000.0),
3   ( 2, 'Bob',     32, 62000.0),
4   ( 3, 'Charlie', 45, 75000.0),
5   ( 4, 'David',   22, 100000.0),
6   ( 5, 'Emily',   36, 80000.0),
7   ( 6, 'Frank',   50, 95000.0),
8   ( 7, 'George',  29, 56000.0),
9   ( 8, 'Henry',   41, 68000.0),
10  ( 9, 'Isabelle', 27, 50000.0),
11  (10, 'Jack',     31, 61000.0);
12
13 INSERT INTO `Dept`(`did`, `dname`, `budget`, `managerid`) VALUES
14   (1, 'Software', 3000000.0, 3),
15   (2, 'Marketing', 75000.0, 5),
16   (3, 'Hardware', 4000001.0, 6),
17   (4, 'Finance',  1000000.0, 7),
18   (5, 'HR',       90000.0, 9),
19   (6, 'Sales',    1000000.0, 6);
20
21 INSERT INTO `Works`(`eid`, `did`, `pct_time`) VALUES
22   ( 1, 1, 100),
23   ( 2, 1, 100),
```



```

24      ( 3, 1, 33),
25      ( 3, 3, 33),
26      ( 3, 4, 34),
27      ( 4, 1, 30),
28      ( 4, 2, 70),
29      ( 5, 1, 30),
30      ( 5, 2, 70),
31      ( 6, 3, 70),
32      ( 6, 6, 30),
33      ( 7, 1, 70),
34      ( 7, 4, 30),
35      ( 8, 1, 70),
36      ( 8, 4, 30),
37      ( 9, 1, 70),
38      ( 9, 5, 30),
39      (10, 1, 70),
40      (10, 5, 30);

```

```

--> ( 4, 'David', 22, 100000.0),
--> ( 5, 'Emily', 30, 80000.0),
--> ( 6, 'Frank', 50, 95000.0),
--> ( 7, 'George', 29, 56000.0),
--> ( 8, 'Henry', 41, 68000.0),
--> ( 9, 'Isabelle', 27, 50000.0),
--> (10, 'Jack', 31, 61000.0);
Query OK, 10 rows affected (0.001 sec)
Records: 10 Duplicates: 0 Warnings: 0

MariaDB [company]>
MariaDB [company]> INSERT INTO `Dept`(`did`, `dname`, `budget`, `managerid`) VALUES
--> (1, 'Software', 3000000.0, 3),
--> (2, 'Marketing', 75000.0, 5),
--> (3, 'Hardware', 4000001.0, 6),
--> (4, 'Finance', 1000000.0, 7),
--> (5, 'HR', 90000.0, 9),
--> (6, 'Sales', 1000000.0, 6);
Query OK, 6 rows affected (0.001 sec)
Records: 6 Duplicates: 0 Warnings: 0

MariaDB [company]>
MariaDB [company]> INSERT INTO `Works`(`eid`, `did`, `pct_time`) VALUES
--> ( 1, 1, 100),
--> ( 2, 1, 100),
--> ( 3, 1, 33),
--> ( 3, 3, 33),
--> ( 3, 4, 34),
--> ( 4, 1, 30),
--> ( 4, 2, 70),
--> ( 5, 1, 30),
--> ( 5, 2, 70),
--> ( 6, 3, 70),
--> ( 6, 6, 30),
--> ( 7, 1, 70),
--> ( 7, 4, 30),
--> ( 8, 1, 70),
--> ( 8, 4, 30),
--> ( 9, 1, 70),
--> ( 9, 5, 30),
--> (10, 1, 70),
--> (10, 5, 30);
Query OK, 19 rows affected (0.000 sec)
Records: 19 Duplicates: 0 Warnings: 0

MariaDB [company]>
MariaDB [company]>

```

1

Print the names and ages of each employee who works in both the Hardware department and the Software department.

```

1 SELECT e.`ename`, e.`age`
2 FROM `Emp` e
3 INNER JOIN `Works` w ON e.`eid` = w.`eid`
4 INNER JOIN `Dept` d ON w.`did` = d.`did`
5 WHERE d.`dname` = 'Hardware'
6 INTERSECT
7 SELECT e.`ename`, e.`age`
8 FROM `Emp` e
9 INNER JOIN `Works` w ON e.`eid` = w.`eid`
10 INNER JOIN `Dept` d ON w.`did` = d.`did`
11 WHERE d.`dname` = 'Software';

```

Ans:

```

1 +-----+-----+
2 | ename   | age |
3 +-----+-----+
4 | Charlie | 45  |
5 +-----+-----+

```

2

For each department with more than 20 full-time-equivalent employees (i.e., where the part-time and full-time employees add up to at least that many full-time employees), print the “did” together with the number of employees that work in that department.

```

1 SELECT `Result`.`did`, `Result`.`num_emp`
2 FROM (
3     SELECT w.`did`, SUM(w.`pct_time`) AS `sum_time`, COUNT(*) AS `num_emp`
4     FROM `Works` w
5     GROUP BY w.`did`
6 ) AS `Result`
7 WHERE `Result`.`sum_time` > 20;

```

Ans:

```

1 +-----+-----+
2 | did | num_emp |
3 +-----+-----+
4 | 1 | 9 |
5 | 2 | 2 |
6 | 3 | 2 |
7 | 4 | 3 |
8 | 5 | 2 |
9 | 6 | 1 |
10 +-----+-----+

```

3

Print the names of each employee whose salary exceeds the budget of all of the departments that he or she works in.

```
1 SELECT e.`ename`
2 FROM `Emp` e
3 INNER JOIN `Works` w ON e.`eid` = w.`eid`
4 INNER JOIN `Dept` d ON w.`did` = d.`did`
5 WHERE e.`salary` > ALL (
6     SELECT d2.`budget`
7     FROM `Works` w2
8     JOIN `Dept` d2 ON w2.`did` = d2.`did`
9     WHERE w2.`eid` = e.`eid`
10 );
```

Ans: Empty set.

4

Find the “managerids” of managers who manage only departments with budgets greater than \$1 million.

```
1 SELECT DISTINCT d.`managerid`
2 FROM `Dept` d
3 WHERE d.`budget` > 1000000
4 AND NOT EXISTS (
5     SELECT 1
6     FROM `Dept` d2
7     WHERE d2.`managerid` = d.`managerid`
8     AND d2.`budget` <= 1000000
9 );
```

Using `SELECT 1` can be a more efficient way to write a subquery that checks for the existence of rows in a table, as the database only needs to return a single constant value instead of all the columns in the table. For example, in the subquery used in the previous answer, we only need to check if there exists any department with a budget less than or equal to \$1 million for a given manager, so we can use `SELECT 1` instead of `SELECT *` to save computation resources.

Ans: 3

5

Find the “enames” of managers who manage the departments with the largest budgets.

```
1 SELECT e.`ename`
2 FROM `Emp` e
3 INNER JOIN `Dept` d ON e.`eid` = d.`managerid`
4 WHERE d.`budget` = (
5     SELECT MAX(`budget`)
6     FROM `Dept`
7 );
```

Ans: Frank

6

If a manager manages more than one department, he or she “controls” the sum of all the budgets for those departments. Find the “managerids” of managers who control more than \$5 million.

```
1 SELECT e.`ename`
2 FROM (
3     SELECT d.`managerid`, SUM(d.`budget`) AS `tot_budget`
4     FROM `Dept` d
5     GROUP BY d.`managerid`
6 ) AS `result`
7 INNER JOIN `Emp` e ON `result`.`managerid` = e.`eid`
8 WHERE `result`.`tot_budget` > 5000000;
```

Ans: Frank

7

Find the “managerids” of managers who control the largest amounts.

```
1 SELECT e.`ename`
2 FROM (
3     SELECT d.`managerid`, SUM(d.`budget`) AS `tot_budget`
4     FROM `Dept` d
5     GROUP BY d.`managerid`
6 ) AS `result`
7 INNER JOIN `Emp` e ON `result`.`managerid` = e.`eid`
8 WHERE `result`.`tot_budget` = (
9     SELECT MAX(`result2`.`tot_budget`)
10    FROM (
11        SELECT d2.`managerid`, SUM(d2.`budget`) AS `tot_budget`
12        FROM `Dept` d2
13        GROUP BY d2.`managerid`
```

```
14 | ) AS `result2`  
15 | );
```

Ans: Frank

8

Find the “enames” of managers who manage only departments with budgets larger than \$1 million, but at least one department with budget less than \$5 million.

```
1 | SELECT DISTINCT e.`ename`  
2 | FROM `Emp` e  
3 | JOIN `Dept` d ON e.`eid` = d.`managerid`  
4 | WHERE d.`budget` > 1000000  
5 | AND EXISTS (  
6 |     SELECT 1  
7 |     FROM `Dept` d2  
8 |     WHERE d2.`managerid` = d.`managerid`  
9 |     AND d2.`budget` < 5000000  
10 | )  
11 | AND NOT EXISTS (  
12 |     SELECT 1  
13 |     FROM `Dept` d3  
14 |     WHERE d3.`managerid` = d.`managerid`  
15 |     AND d3.`budget` <= 1000000  
16 | );
```

Ans: Charlie

Part C

DDL

```
1 | CREATE DATABASE IF NOT EXISTS `course_2`;  
2 | USE `course_2`;  
3 |  
4 | -- Entities  
5 | CREATE TABLE IF NOT EXISTS `Students` (  
6 |     `sid` INT NOT NULL PRIMARY KEY,  
7 |     `sname` TEXT NOT NULL,  
8 |     `age` INT NOT NULL  
9 | );  
10 |  
11 | CREATE TABLE IF NOT EXISTS `Courses` (  
12 |     `cid` INT NOT NULL PRIMARY KEY,  
13 |     `cname` TEXT NOT NULL,  
14 |     `credits` INT NOT NULL
```

```

15 );
16
17 -- Relations
18 CREATE TABLE IF NOT EXISTS `Enrolled` (
19     `sid` INT NOT NULL,
20     `cid` INT NOT NULL,
21     `grade` INT NOT NULL,
22     PRIMARY KEY (`sid`, `cid`),
23     FOREIGN KEY (`sid`)
24         REFERENCES `Students`(`sid`)
25         ON DELETE CASCADE
26         ON UPDATE CASCADE,
27     FOREIGN KEY (`cid`)
28         REFERENCES `Courses`(`cid`)
29         ON DELETE CASCADE
30         ON UPDATE CASCADE
31 );

```

The screenshot shows a MariaDB terminal window on the left and a SQL editor on the right. The terminal output shows the following commands and results:

```

Server version: 10.11.2-MariaDB Arch Linux
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> CREATE DATABASE IF NOT EXISTS `course_2`;
Query OK, 1 row affected (0.000 sec)

MariaDB [(none)]> USE `course_2`;
Database changed
MariaDB [course_2]> -- Entities
MariaDB [course_2]> CREATE TABLE IF NOT EXISTS `Students` (
  -> `sid` INT NOT NULL PRIMARY KEY,
  -> `sname` TEXT NOT NULL,
  -> `age` INT NOT NULL
  -> );
Query OK, 0 rows affected (0.003 sec)

MariaDB [course_2]>
MariaDB [course_2]> CREATE TABLE IF NOT EXISTS `Courses` (
  -> `cid` INT NOT NULL PRIMARY KEY,
  -> `cname` TEXT NOT NULL,
  -> `credits` INT NOT NULL
  -> );
Query OK, 0 rows affected (0.002 sec)

MariaDB [course_2]>
MariaDB [course_2]> -- Relations
MariaDB [course_2]> CREATE TABLE IF NOT EXISTS `Enrolled` (
  -> `sid` INT NOT NULL,
  -> `cid` INT NOT NULL,
  -> `grade` INT NOT NULL,
  -> PRIMARY KEY (`sid`, `cid`),
  -> FOREIGN KEY (`sid`)
  -> REFERENCES `Students`(`sid`)
  -> ON DELETE CASCADE
  -> ON UPDATE CASCADE,
  -> FOREIGN KEY (`cid`)
  -> REFERENCES `Courses`(`cid`)
  -> ON DELETE CASCADE
  -> ON UPDATE CASCADE
  -> );
Query OK, 0 rows affected (0.004 sec)

MariaDB [course_2]>

```

The SQL editor on the right shows the same SQL code as the first block, with the `Enrolled` table definition highlighted.

DML

```

1 INSERT INTO `Students`(`sid`, `sname`, `age`) VALUES
2     ( 1, 'Alice', 18),
3     ( 2, 'Bob', 25),
4     ( 3, 'Charlie', 20),
5     ( 4, 'David', 19),
6     ( 5, 'Ella', 23),
7     ( 6, 'Frank', 24),
8     ( 7, 'Grace', 18),
9     ( 8, 'Henry', 22),
10    ( 9, 'Ivy', 25),

```

```

11      (10, 'John',    21);
12
13  INSERT INTO `Courses`(`cid`, `cname`, `credits`) VALUES
14      (1, 'Math',      4),
15      (2, 'Science',   3),
16      (3, 'English',   4),
17      (4, 'Databases', 3),
18      (5, 'Art',        2),
19      (6, 'Music',      2),
20      (7, 'Computer Sci', 4);
21
22  INSERT INTO `Enrolled`(`sid`, `cid`, `grade`) VALUES
23      ( 1, 5,  6),
24      ( 1, 7, 10),
25      ( 2, 7,  7),
26      ( 3, 1,  7),
27      ( 3, 3,  7),
28      ( 3, 7,  9),
29      ( 4, 1,  9),
30      ( 4, 7,  8),
31      ( 5, 3,  7),
32      ( 5, 7,  7),
33      ( 6, 4, 10),
34      ( 6, 6,  8),
35      ( 7, 5,  8),
36      ( 7, 7,  9),
37      ( 8, 1,  6),
38      ( 8, 7,  8),
39      ( 9, 2,  7),
40      ( 9, 4,  7),
41      (10, 7,  8);

```

The screenshot shows a Visual Studio Code editor with a MySQL console window on the left and a file explorer on the right. The console window displays the following SQL commands and their results:

```

MariaDB [(none)]> USE `course_2`;
Query OK, 0 rows affected (0.001 sec)

MariaDB [(none)]> INSERT INTO `Courses`(`cid`, `cname`, `credits`) VALUES
(1, 'Math', 4),
(2, 'Science', 3),
(3, 'English', 4),
(4, 'Databases', 3),
(5, 'Art', 2),
(6, 'Music', 2),
(7, 'Computer Sci', 4);
Query OK, 7 rows affected (0.001 sec)
Records: 7 Duplicates: 0 Warnings: 0

MariaDB [(none)]> INSERT INTO `Enrolled`(`sid`, `cid`, `grade`) VALUES
(1, 5, 6),
(1, 7, 10),
(2, 7, 7),
(3, 1, 7),
(3, 3, 7),
(3, 7, 9),
(4, 1, 9),
(4, 7, 8),
(5, 3, 7),
(5, 7, 7),
(6, 4, 10),
(6, 6, 8),
(7, 5, 8),
(7, 7, 9),
(8, 1, 6),
(8, 7, 8),
(9, 2, 7),
(9, 4, 7),
(10, 7, 8);
Query OK, 19 rows affected (0.001 sec)
Records: 19 Duplicates: 0 Warnings: 0

```

The file explorer on the right shows a project structure with various files and folders, including a 'part_CDDL.sql' file which contains the SQL commands shown in the console window.

1

Write a statement to create the table **Enrolled**. You do **not** need to provide create table statements for the other tables. Include necessary key constraints.

```
1 CREATE TABLE IF NOT EXISTS `Enrolled` (  
2     `sid` INT NOT NULL,  
3     `cid` INT NOT NULL,  
4     `grade` INT NOT NULL,  
5     PRIMARY KEY (`sid`, `cid`),  
6     FOREIGN KEY (`sid`)  
7         REFERENCES `Students`(`sid`)  
8         ON DELETE CASCADE  
9         ON UPDATE CASCADE,  
10    FOREIGN KEY (`cid`)  
11        REFERENCES `Courses`(`cid`)  
12        ON DELETE CASCADE  
13        ON UPDATE CASCADE  
14 );
```

2

Find the name(s) of student(s) with the youngest age.

```
1 SELECT s.`sname`  
2 FROM `Students` s  
3 WHERE s.`age` = (  
4     SELECT MIN(s2.`age`)  
5     FROM `Students` s2  
6 );
```

Ans: Alice and Grace.

3

Find the ages of students who take only courses with less than four credits (implies they take at least one course).

```
1 SELECT `Result`.`age`  
2 FROM (  
3     SELECT DISTINCT s.`sid`, s.`age`  
4     FROM `Students` s  
5     INNER JOIN `Enrolled` e ON s.`sid` = e.`sid`  
6     WHERE e.`cid` IN (  
7         SELECT e2.`cid`  
8         FROM `Enrolled` e2  
9         WHERE e2.`credits` < 4  
10    )  
11 )
```



```

9      ) AND s.`sid` NOT IN (
10         SELECT e3.`sid`
11         FROM `Enrolled` e3
12         INNER JOIN `Courses` c ON e3.`cid` = c.`cid`
13         WHERE c.`credits` >= 4
14     )
15 ) AS `Result`;

```

Ans: 24 and 25.

4

Find the ages of students who got grade 10 in a course named 'Databases'.

```

1  SELECT s.`age`
2  FROM `Students` s
3  INNER JOIN `Enrolled` e ON s.`sid` = e.`sid`
4  INNER JOIN `Courses` c ON e.`cid` = c.`cid`
5  WHERE e.`grade` = 10 AND c.`cname` = 'Databases';

```

Ans: 24

5

Find the course identifier cid and the average age over enrolled students who are 20 or older for each course that has at least 50 enrolled students (of any age).

```

1  SELECT e.`cid`, AVG(s.`age`) AS `students_avg_age`
2  FROM `Enrolled` e
3  INNER JOIN `Students` s ON e.`sid` = s.`sid`
4  GROUP BY e.`cid`
5  HAVING COUNT(*) >= 50;

```

Ans: Empty Set.

6

Find the names of students who take all the four-credit courses offered and obtained at least grade 7 in every such course.

```

1  SELECT s.`sname`
2  FROM `Students` s
3  INNER JOIN `Enrolled` e ON s.`sid` = e.`sid`
4  INNER JOIN `Courses` c ON e.`cid` = c.`cid`
5  WHERE c.`credits` = 4 AND e.`grade` >= 7
6  GROUP BY s.`sid`
7  HAVING COUNT(DISTINCT e.`cid`) = (
8      SELECT COUNT(*)
9      FROM `Courses`
10     WHERE `credits` = 4
11 );

```

Ans: Charlie.

7

Find the name(s) of students(s) who have the highest GPA (assume the GPA is computed only based on grades available in **Enrolled**).

```

1  SELECT s.`sname`
2  FROM (
3      SELECT *, `weight_sum_grade` / `cnt_courses` AS `GPA`
4      FROM (
5          SELECT r.`sid`, SUM(r.`weight_grade`) AS `weight_sum_grade`,
6              COUNT(r.`sid`) AS `cnt_courses`
7          FROM (
8              SELECT e.`sid`, e.`grade` * c.`credits` AS `weight_grade`
9              FROM `Enrolled` e
10             INNER JOIN `Courses` c ON e.`cid` = c.`cid`
11          ) AS r
12          GROUP BY r.`sid`
13      ) AS r2
14  ) AS r3
15  INNER JOIN `Students` s ON r3.`sid` = s.`sid`
16  WHERE r3.`GPA` = (
17      SELECT MAX(r6.`GPA`)
18      FROM (
19          SELECT *, `weight_sum_grade` / `cnt_courses` AS `GPA`
20          FROM (
21              SELECT r4.`sid`, SUM(r4.`weight_grade`) AS `weight_sum_grade`,
22                  COUNT(r4.`sid`) AS `cnt_courses`
23              FROM (
24                  SELECT e2.`sid`, e2.`grade` * c2.`credits` AS `weight_grade`
25                  FROM `Enrolled` e2
26                  INNER JOIN `Courses` c2 ON e2.`cid` = c2.`cid`
27              ) AS r4
28              GROUP BY r4.`sid`
29          ) AS r5
30      ) AS r6

```

```
31 | );
```

Ans: David.

8

Find the ages of students who take some course with 3 credits.

```
1  SELECT `Result`.`age`  
2  FROM (  
3      SELECT DISTINCT s.`sid`, s.`age`  
4      FROM `Students` s  
5      INNER JOIN `Enrolled` e ON s.`sid` = e.`sid`  
6      WHERE e.`cid` IN (  
7          SELECT c.`cid`  
8          FROM `Courses` c  
9          WHERE c.`credits` = 3  
10     )  
11 ) AS `Result`;
```

Ans: 24, 25.

9

Find the names of students who obtained grade at least 8 in some course that has less than 4 credits.

```
1  SELECT `Result`.`sname`  
2  FROM (  
3      SELECT DISTINCT s.`sid`, s.`sname`  
4      FROM `Students` s  
5      INNER JOIN `Enrolled` e ON s.`sid` = e.`sid`  
6      WHERE e.`cid` IN (  
7          SELECT c.`cid`  
8          FROM `Courses` c  
9          WHERE c.`credits` < 4  
10     ) AND e.`grade` >= 8  
11 ) AS `Result`;
```

Ans: Frank, Grace.

10

Find the names of students who obtained only grades of 10 (implies that they took at least one course).

```
1 SELECT s.`sname`
2 FROM `Students` s
3 INNER JOIN `Enrolled` e ON s.`sid` = e.`sid`
4 WHERE NOT EXISTS (
5     SELECT 1
6     FROM `Enrolled` e2
7     WHERE e.`sid` = e2.`sid`
8     AND e2.`grade` <> 10
9 );
```

Ans: Empty Set.

11

Find the names of students who took a course with three credits or who obtained grade 10 in some course.

```
1 SELECT `Result`.`sname`
2 FROM (
3     SELECT DISTINCT s.`sid`, s.`sname`
4     FROM `Students` s
5     INNER JOIN `Enrolled` e ON s.`sid` = e.`sid`
6     WHERE e.`grade` = 10
7     UNION
8     SELECT DISTINCT s2.`sid`, s2.`sname`
9     FROM `Students` s2
10    INNER JOIN `Enrolled` e2 ON s2.`sid` = e2.`sid`
11    WHERE e2.`cid` IN (
12        SELECT c.`cid`
13        FROM `Courses` c
14        WHERE c.`credits` = 3
15    )
16 ) AS `Result`;
```

Ans: Alice, Frank, Ivy.

12

Find the names of students who are enrolled in a single course.

```
1 SELECT s.`sname`
2 FROM `Students` s
3 INNER JOIN (
4     SELECT e.`sid`, COUNT(*) AS `cnt_courses`
5     FROM `Enrolled` e
6     GROUP BY e.`sid`
7     HAVING `cnt_courses` = 1
8 ) AS r ON s.`sid` = r.`sid`;
```

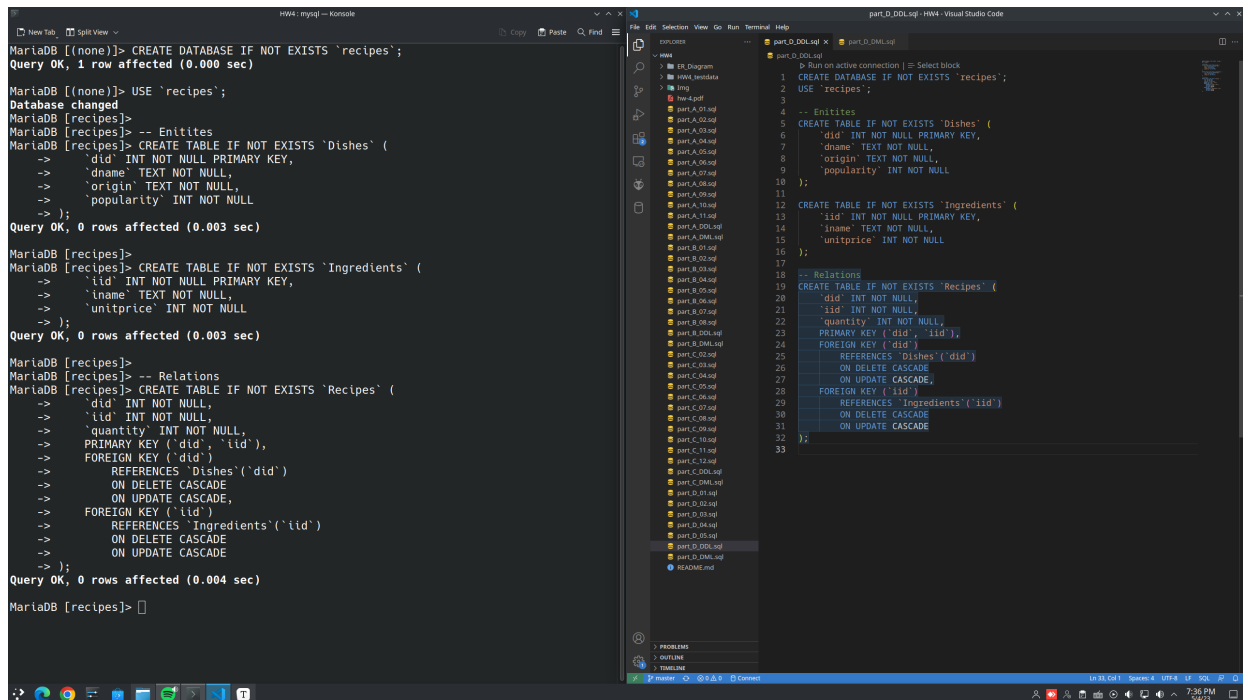
Ans: Bob, John.

Part D

DDL

```
1 CREATE DATABASE IF NOT EXISTS `recipes`;
2 USE `recipes`;
3
4 -- Entities
5 CREATE TABLE IF NOT EXISTS `Dishes` (
6     `did` INT NOT NULL PRIMARY KEY,
7     `dname` TEXT NOT NULL,
8     `origin` TEXT NOT NULL,
9     `popularity` INT NOT NULL
10 );
11
12 CREATE TABLE IF NOT EXISTS `Ingredients` (
13     `iid` INT NOT NULL PRIMARY KEY,
14     `iname` TEXT NOT NULL,
15     `unitprice` INT NOT NULL
16 );
17
18 -- Relations
19 CREATE TABLE IF NOT EXISTS `Recipes` (
20     `did` INT NOT NULL,
21     `iid` INT NOT NULL,
22     `quantity` INT NOT NULL,
23     PRIMARY KEY (`did`, `iid`),
24     FOREIGN KEY (`did`)
25         REFERENCES `Dishes`(`did`)
26         ON DELETE CASCADE
27         ON UPDATE CASCADE,
28     FOREIGN KEY (`iid`)
29         REFERENCES `Ingredients`(`iid`)
30         ON DELETE CASCADE
```

```
31 ON UPDATE CASCADE
32 );
```



```
part_D_DDL.sql
1 CREATE DATABASE IF NOT EXISTS 'recipes';
2 USE 'recipes';
3
4 -- Entities
5 CREATE TABLE IF NOT EXISTS 'Dishes' (
6   'did' INT NOT NULL PRIMARY KEY,
7   'dname' TEXT NOT NULL,
8   'origin' TEXT NOT NULL,
9   'popularity' INT NOT NULL
10 );
11
12 CREATE TABLE IF NOT EXISTS 'Ingredients' (
13   'iid' INT NOT NULL PRIMARY KEY,
14   'iname' TEXT NOT NULL,
15   'unitprice' INT NOT NULL
16 );
17
18 -- Relations
19 CREATE TABLE IF NOT EXISTS 'Recipes' (
20   'did' INT NOT NULL,
21   'iid' INT NOT NULL,
22   'quantity' INT NOT NULL,
23   PRIMARY KEY ('did', 'iid'),
24   FOREIGN KEY ('did')
25     REFERENCES 'Dishes' ('did')
26     ON DELETE CASCADE
27     ON UPDATE CASCADE,
28   FOREIGN KEY ('iid')
29     REFERENCES 'Ingredients' ('iid')
30     ON DELETE CASCADE
31     ON UPDATE CASCADE
32 );
33
```

DML

```
1 INSERT INTO `Dishes`(`did`, `dname`, `origin`, `popularity`) VALUES
2   (1, 'Spaghetti', 'Italy', 8),
3   (2, 'Sushi', 'Japan', 9),
4   (3, 'Tacos', 'Mexico', 7),
5   (4, 'Paella', 'Spain', 6),
6   (5, 'Pad Thai', 'Thailand', 8),
7   (6, 'Burger', 'USA', 10001);
8
9 INSERT INTO `Ingredients`(`iid`, `iname`, `unitprice`) VALUES
10  ( 1, 'Pasta', 2),
11  ( 2, 'Tomatoes', 3),
12  ( 3, 'saffron', 50),
13  ( 4, 'Rice', 1),
14  ( 5, 'sugar', 4),
15  ( 6, 'butter', 6),
16  ( 7, 'Beef', 55),
17  ( 8, 'Chicken', 7),
18  ( 9, 'starch', 4),
19  (10, 'Potatoes', 2);
20
21 INSERT INTO `Recipes`(`did`, `iid`, `quantity`) VALUES
22  (1, 1, 200),
23  (1, 2, 50),
24  (1, 3, 100),
25  (2, 5, 10),
```

```

26      (2, 6, 20),
27      (3, 2, 30),
28      (3, 8, 50),
29      (4, 4, 100),
30      (5, 1, 150),
31      (5, 2, 30),
32      (5, 9, 50),
33      (6, 7, 50);

```

The screenshot shows a MySQL terminal window on the left and a Visual Studio Code editor on the right. The terminal displays three SQL queries and their results:

```

MariaDB [recipes]> INSERT INTO `Dishes` (`did`, `dname`, `origin`, `popularity`) VALUES
-> (1, 'Spaghetti', 'Italy', 8),
-> (2, 'Sushi', 'Japan', 9),
-> (3, 'Tacos', 'Mexico', 7),
-> (4, 'Paella', 'Spain', 6),
-> (5, 'Pad Thai', 'Thailand', 8),
-> (6, 'Burger', 'USA', 10001);
Query OK, 6 rows affected (0.001 sec)
Records: 6 Duplicates: 0 Warnings: 0

MariaDB [recipes]> INSERT INTO `Ingredients` (`iid`, `iname`, `unitprice`) VALUES
-> (1, 'Pasta', 2),
-> (2, 'Tomatoes', 3),
-> (3, 'saffron', 50),
-> (4, 'Rice', 1),
-> (5, 'sugar', 4),
-> (6, 'butter', 6),
-> (7, 'Beef', 55),
-> (8, 'Chicken', 7),
-> (9, 'starch', 4),
-> (10, 'Potatoes', 2);
Query OK, 10 rows affected (0.000 sec)
Records: 10 Duplicates: 0 Warnings: 0

MariaDB [recipes]> INSERT INTO `Recipes` (`did`, `iid`, `quantity`) VALUES
-> (1, 1, 200),
-> (1, 2, 50),
-> (1, 3, 100),
-> (2, 5, 10),
-> (2, 6, 20),
-> (3, 2, 30),
-> (3, 8, 50),
-> (4, 4, 100),
-> (5, 1, 150),
-> (5, 2, 30),
-> (5, 9, 50),
-> (6, 7, 50);
Query OK, 12 rows affected (0.000 sec)
Records: 12 Duplicates: 0 Warnings: 0

```

The Visual Studio Code editor on the right shows a file named 'part_D_DML.sql' with the following SQL script:

```

-- Run on active connection | = Select block
1  USE `recipes`;
2
3  -- Delete all rows for every tables.
4  DELETE FROM `Dishes`;
5  DELETE FROM `Ingredients`;
6  DELETE FROM `Recipes`;
7
8  -- These are from sample data.
9  INSERT INTO `Dishes` (`did`, `dname`, `origin`, `popularity`) VALUES
10 (1, 'Spaghetti', 'Italy', 8),
11 (2, 'Sushi', 'Japan', 9),
12 (3, 'Tacos', 'Mexico', 7),
13 (4, 'Paella', 'Spain', 6),
14 (5, 'Pad Thai', 'Thailand', 8),
15 (6, 'Burger', 'USA', 10001);
16
17 INSERT INTO `Ingredients` (`iid`, `iname`, `unitprice`) VALUES
18 (1, 'Pasta', 2),
19 (2, 'Tomatoes', 3),
20 (3, 'saffron', 50),
21 (4, 'Rice', 1),
22 (5, 'sugar', 4),
23 (6, 'butter', 6),
24 (7, 'Beef', 55),
25 (8, 'Chicken', 7),
26 (9, 'starch', 4),
27 (10, 'Potatoes', 2);
28
29 INSERT INTO `Recipes` (`did`, `iid`, `quantity`) VALUES
30 (1, 1, 200),
31 (1, 2, 50),
32 (1, 3, 100),
33 (2, 5, 10),
34 (2, 6, 20),
35 (3, 2, 30),
36 (3, 8, 50),
37 (4, 4, 100),
38 (5, 1, 150),
39 (5, 2, 30),
40 (5, 9, 50),
41 (6, 7, 50);
42

```

1

Find the dish names that do NOT contain any of the following ingredients: sugar, butter, starch.

```

1  SELECT `Result`.`dname`
2  FROM (
3      SELECT DISTINCT d.`did`, d.`dname`
4      FROM `Dishes` d
5      INNER JOIN `Recipes` r ON d.`did` = r.`did`
6      INNER JOIN `Ingredients` i ON r.`iid` = i.`iid`
7      WHERE i.`iname` <> 'sugar'
8      AND i.`iname` <> 'butter'
9      AND i.`iname` <> 'starch'
10 ) AS `Result`;

```

Ans: Spaghetti, Tacos, Paella, Pad Thai, Burger.

2

Find the ingredient names that cost at least \$10 per unit and that appear in at least one dish with popularity higher than 10,000.

```
1 SELECT `Result`.`iname`
2 FROM (
3     SELECT DISTINCT i.`iid`, i.`iname`
4     FROM `Ingredients` i
5     INNER JOIN `Recipes` r ON i.`iid` = r.`iid`
6     INNER JOIN `Dishes` d ON r.`did` = d.`did`
7     WHERE i.`unitprice` >= 10
8     AND d.`popularity` > 10000
9 ) AS `Result`;
```

Ans: Beef.

3

Find the origin of dishes that use at least one unit of an ingredient called 'saffron'.

```
1 SELECT `Result`.`origin`
2 FROM (
3     SELECT DISTINCT d.`did`, d.`origin`
4     FROM `Dishes` d
5     INNER JOIN `Recipes` r ON d.`did` = r.`did`
6     INNER JOIN `Ingredients` i ON r.`iid` = i.`iid`
7     WHERE i.`iname` = 'saffron'
8     AND r.`quantity` >= 1
9 ) AS `Result`;
```

Ans: Italy.

4

List the popularity of “exclusive” dishes, defined as dishes that contain only ingredients costing at least \$50 per unit.

```
1 SELECT `Result`.`popularity`
2 FROM (
3     SELECT d.`did`, d.`popularity`
4     FROM `Dishes` d
5     WHERE NOT EXISTS (
6         SELECT 1
7         FROM `Recipes` r
8         WHERE d.`did` = r.`did`
9         AND EXISTS (
```



```

10         SELECT 1
11         FROM `Ingredients` i
12         WHERE r.`iid` = i.`iid`
13         AND i.`unitprice` < 50
14     )
15 )
16 ) AS `Result`;

```

Ans: 10001.

5

Find the name and unit price of rare ingredients, i.e., those that appear in a single dish.

```

1 SELECT i.`iname`, i.`unitprice`
2 FROM `Ingredients` i
3 INNER JOIN (
4     SELECT r.`iid`, COUNT(*) AS `cnt_dishes`
5     FROM `Recipes` r
6     GROUP BY r.`iid`
7     HAVING `cnt_dishes` = 1
8 ) AS `Result` ON i.`iid` = `Result`.`iid`;

```

Ans:

```

1  +-----+-----+
2  | iname   | unitprice |
3  +-----+-----+
4  | saffron |         50 |
5  | Rice    |          1 |
6  | sugar   |          4 |
7  | butter  |          6 |
8  | Beef    |         55 |
9  | Chicken |          7 |
10 | starch  |          4 |
11 +-----+-----+

```