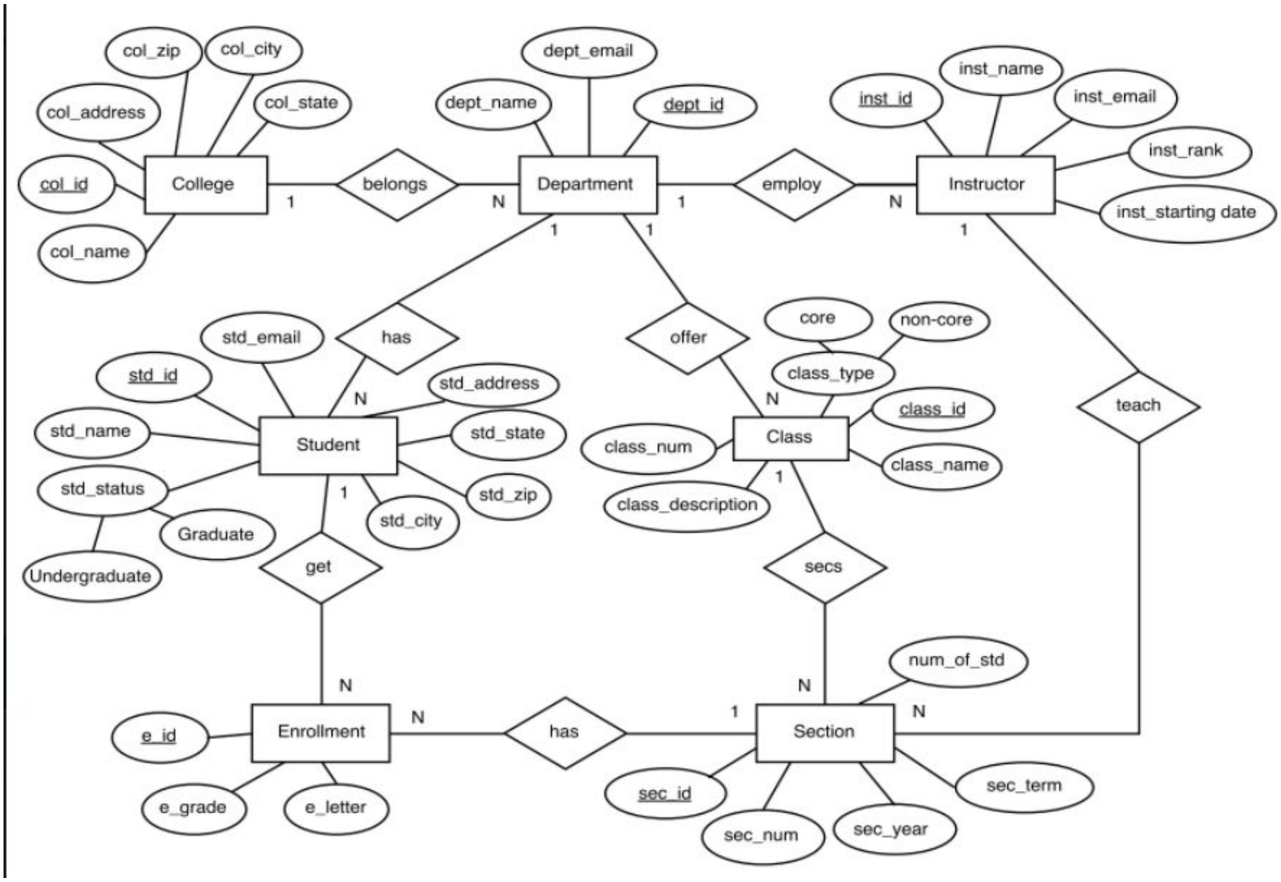
1. Exercise 1 : ER diagram



2. Exercise 2 : Relational Model

```
CREATE DATABASE IF NOT EXISTS university;

USE university;

CREATE TABLE college (
 col_id INTEGER PRIMARY KEY AUTO_INCREMENT,
 col_name VARCHAR(100) NOT NULL,
 col_address VARCHAR(200) NOT NULL,
 col_state VARCHAR(20) NOT NULL,
 col_city VARCHAR(100) NOT NULL,
 col_zip VARCHAR(10) NOT NULL
);

CREATE TABLE department (
 dept_id INTEGER PRIMARY KEY AUTO_INCREMENT,
 dept_name VARCHAR(500) NOT NULL,
 dept_email VARCHAR(200) NOT NULL,
 col_id INTEGER NOT NULL,

 FOREIGN KEY (col_id) REFERENCES college (col_id),
 INDEX (col_id)
);

CREATE TABLE instructor (
 inst_id INTEGER PRIMARY KEY AUTO_INCREMENT,
 inst_name VARCHAR(500) NOT NULL,
 inst_email VARCHAR(200) NOT NULL,
 inst_rank ENUM ('Lecture', 'Sr.Lecturer', 'Assistant',
'associate', 'Full-Professor'),
 inst_starting_date DATE NOT NULL,
 dept_id INTEGER NOT NULL,

 FOREIGN KEY (dept_id) REFERENCES department (dept_id),
 INDEX (dept_id)
);

CREATE TABLE student (
 std_id INTEGER PRIMARY KEY AUTO_INCREMENT,
 std_name VARCHAR(500) NOT NULL,
 std_email VARCHAR(200) NOT NULL,
 std_status ENUM ('Undergraduate', 'Graduate'),
 std_address VARCHAR(200) NOT NULL,
 std_state VARCHAR(20) NOT NULL,
 std_city VARCHAR(100) NOT NULL,
 std_zip VARCHAR(10) NOT NULL,
 dept_id INTEGER NOT NULL,

 FOREIGN KEY (dept_id) REFERENCES department (dept_id),
 INDEX (dept_id)
);

CREATE TABLE enrollment (
 e_id INTEGER PRIMARY KEY AUTO_INCREMENT,
 e_grade DECIMAL NOT NULL,
 e_letter ENUM ('A', 'B', 'C', 'D', 'F'),
 std_id INTEGER NOT NULL,
 sec_id INTEGER NOT NULL,

 FOREIGN KEY (std_id) REFERENCES student (std_id),
 FOREIGN KEY (sec_id) REFERENCES section (sec_id),
 INDEX (std_id),
 INDEX (sec_id)
);

CREATE TABLE class (
 class_id INTEGER PRIMARY KEY AUTO_INCREMENT,
 class_num INTEGER NOT NULL,
 class_name VARCHAR(500) NOT NULL,
 class_description VARCHAR(1000) NOT NULL,
 class_type ENUM ('Core', 'Non-core'),
 dept_id INTEGER NOT NULL,

 FOREIGN KEY (dept_id) REFERENCES department (dept_id),
 INDEX (dept_id)
);

CREATE TABLE section (
 sec_id INTEGER PRIMARY KEY AUTO_INCREMENT,
 sec_num INTEGER NOT NULL,
 sec_year YEAR(4) NOT NULL,
 sec_term ENUM ('Spring', 'Fall', 'Summer'),
 num_of_std INT NOT NULL,
 class_id INTEGER NOT NULL REFERENCES class,
 inst_id INTEGER NOT NULL REFERENCES instructor,

 FOREIGN KEY (class_id) REFERENCES class (class_id),
 FOREIGN KEY (inst_id) REFERENCES instructor (inst_id),
 INDEX (class_id),
 INDEX (inst_id)
);
```

3. Exercise 3 : Queries in SQL

a. Retrieve the list of all core classes.

```
SELECT c.dept_id, dept_name, class_id, class_num,
class_name, class_type
FROM department
JOIN class c ON department.dept_id = c.dept_id
WHERE class_type = 'Core'
ORDER BY dept_name, class_num;
```

b. StudentID of students who live in Boise and tool CS410, in ascending order.

```
SELECT e.std_id, std_name, std_city, sec_num, class_name
FROM student
 JOIN enrollment e ON student.std_id = e.std_id
 JOIN section s ON e.sec_id = s.sec_num
 JOIN class c ON s.class_id = c.class_num
WHERE std_city = 'Boise'
 AND class_name = 'Databases'
 AND e_grade IS NOT NULL
GROUP BY std_id, class_name, sec_num
ORDER BY std_name;
```

c. Display details of Undergraduate students with cumulative GPA greater than 3.8. Compute the cumulative GPA as the average of all the grades got by a student.

```
SELECT e.std_id, std_name, std_email, std_address,
std_city, std_state, std_zip, std_status, AVG(e_grade) AS
'GPA'
FROM student
JOIN enrollment e ON student.std_id = e.std_id
WHERE std_status = 'Undergraduate'
 AND e_grade IS NOT NULL
GROUP BY e.std_id
HAVING GPA > 3.8
ORDER BY GPA DESC;
```

d. Retrieve StudentID and Name of students who have taken ALL the core classes.

```
SELECT s.std_id, s.std_name, class_type
FROM department
 JOIN student s ON department.dept_id = s.dept_id
 JOIN enrollment r ON s.std_id = r.std_id
 LEFT JOIN section s2 ON r.sec_id = s2.sec_id
 JOIN class c ON s2.class_id = c.class_id
WHERE s.std_id IN ( SELECT r.e_grade FROM department
                   JOIN student s ON department.dept_id =
s.dept_id
```

```
                   JOIN enrollment r ON s.std_id = r.std_id
                   JOIN section s2 ON r.sec_id = s2.sec_id
                   JOIN class c ON s2.class_id = c.class_id
                   WHERE s.dept_id = c.dept_id
                   AND c.class_type = 'Core')
AND r.e_grade IS NOT NULL
GROUP BY  s.std_id, class_type;
```

e. Retrieve details of Instructors who taught more than 3 core classes.

```
SELECT s.inst_id, inst_name, inst_rank, inst_email,
inst_starting_date,c.dept_id, dept_name, COUNT(sec_id) AS
num_of_coreclass
FROM department
 JOIN instructor i ON department.dept_id = i.dept_id
 JOIN class c ON department.dept_id = c.dept_id
 JOIN section s ON i.inst_id = s.inst_id
WHERE class_type = 'Core'
GROUP BY i.inst_id
HAVING num_of_coreclass > 3
ORDER BY dept_name, num_of_coreclass DESC;
```

f. Provide Names of the Departments with the highest number of Undergraduate students.

```
SELECT s2.dept_id, dept_name, dept_email, COUNT(std_id) AS
'num_under'
FROM department
 JOIN student s2 ON department.dept_id = s2.dept_id
 JOIN class c ON department.dept_id = c.dept_id
 JOIN section s ON c.class_id = s.class_id
WHERE std_status = 'Undergraduate'
GROUP BY dept_id
ORDER BY num_under DESC;
```

g. Provide Names of the Departments with least number of graduate students.

```
SELECT s.dept_id, dept_name, dept_email, COUNT(std_id) As
'num_grad_std'
```

```
FROM department
 JOIN student s ON department.dept_id = s.dept_id
WHERE std_status = 'Graduate'
GROUP BY s.dept_id
ORDER BY num_grad_std ASC;
```

h. List of Colleges with more than 10 Departments in descending order.

```
SELECT d.col_id, col_name, COUNT(dept_id) AS 'num_of_dept'
FROM college
 JOIN department d ON college.col_id = d.col_id
GROUP BY d.col_id
HAVING COUNT(dept_id) > 10
ORDER BY col_name DESC;
```

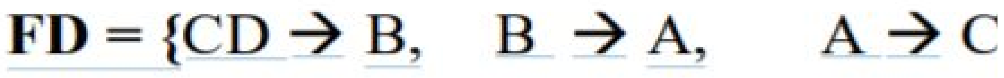
i. Details of all Instructors who taught more than 100 students total.

```
SELECT s.inst_id, inst_name, inst_email, inst_rank,
inst_starting_date, COUNT(s2.std_id) AS 'num_of_std'
FROM instructor
 JOIN section s ON instructor.inst_id = s.inst_id
 JOIN class c ON s.class_id = c.class_id
 JOIN enrollment e ON s.sec_id = e.sec_id
 JOIN student s2 ON e.std_id = s2.std_id
GROUP BY s.inst_id
HAVING num_of_std >= 100
ORDER BY num_of_std;
```

j. The average number of students per semester who tool CS121. Note that there are multiple sections of 121 each semester!

```
SELECT s.sec_term, s.sec_year, avg(x.stunum) AS
Avg_num_students
FROM class class
 JOIN section s ON class.class_id = s.class_id
 JOIN enrollment g ON s.sec_id = g.sec_id
 JOIN student s2 ON g.std_id = s2.std_id
 JOIN (SELECT count(s2.std_id) AS stunum
       FROM class class
       JOIN section s ON class.class_id = s.class_id
       JOIN enrollment g ON s.sec_id = g.sec_id
       JOIN student s2 ON g.std_id = s2.std_id
    WHERE class_name = 'CS121'
    GROUP BY class.class_id, s.sec_year, s.sec_term
    HAVING stunum IS NOT NULL ) AS x
WHERE class_name = 'CS121'
GROUP BY s.sec_year,s.sec_term
ORDER BY s.sec_year, s.sec_term;
```

4. Exercise 4



$FD = \{CD \rightarrow B, \quad B \rightarrow A, \quad A \rightarrow C$

a. Find the set of keys for this relations.

b. Is the relation R in Boyce-Codd Normal Form(BCNF)?

: Yes

c. Is the relation R in Third Normal Form(3NF)? If not, decompose R in 3NF.

: No. Because if no non prime attribute is transitively dependent on the primary key. Each row need to describe just the entity, not related entities.