Hw 5 Solution

```
-- Part 1
CREATE TABLE PROFESSOR (
    ID              int         NOT NULL,   -- the primary key, so not null
    first           varchar(50),            -- make this big enough to hold most name
s
    last            varchar(50),
    PRIMARY KEY (ID)
)   CHARACTER SET 'utf8mb4'
    COLLATE 'utf8mb4_unicode_520_ci';

CREATE TABLE STUDENT (
    ID              int         NOT NULL,   -- the primary key, so not null
    first           varchar(50),            -- make this big enough to hold most name
s
    last            varchar(50),
    major           varchar(50),
    PRIMARY KEY (ID)
)   CHARACTER SET 'utf8mb4'
    COLLATE 'utf8mb4_unicode_520_ci';

CREATE TABLE COURSE (
    catnum          varchar(20) NOT NULL,   -- the primary key, so not null
    description     varchar(2000) NOT NULL, -- this was a candidate key, so we know i
t's not null
    ge_area         varchar(10),
    units           int,
    PRIMARY KEY (catnum),
    UNIQUE (description)                    -- this was a candidate key, so we know i
t's unique
)   CHARACTER SET 'utf8mb4'
    COLLATE 'utf8mb4_unicode_520_ci';

CREATE TABLE PROF_EMAIL (
    ID              int          NOT NULL,  -- part of the primary key, so not null
    email           varchar(255) NOT NULL,  -- part of the primary key, so not null
    PRIMARY KEY (ID, email),
    FOREIGN KEY (ID) REFERENCES PROFESSOR(ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
)   CHARACTER SET 'utf8mb4'
    COLLATE 'utf8mb4_unicode_520_ci';

CREATE TABLE STUDENT_EMAIL (
    ID              int          NOT NULL,  -- part of the primary key, so not null
    email           varchar(255) NOT NULL,  -- part of the primary key, so not null
    PRIMARY KEY (ID, email),
    FOREIGN KEY (ID) REFERENCES STUDENT(ID)
```

```
        ON DELETE CASCADE
        ON UPDATE CASCADE
)   CHARACTER SET 'utf8mb4'
    COLLATE 'utf8mb4_unicode_520_ci';

CREATE TABLE SECTION (
    catnum          varchar(20)  NOT NULL,   -- part of the primary key, so not null
    sectnum         int          NOT NULL,   -- part of the primary key, so not null
    semester        varchar(20)  NOT NULL,   -- part of the primary key, so not null
    room_num        varchar(20),
    prof_ID         int,
    PRIMARY KEY (catnum, sectnum, semester),
    FOREIGN KEY (catnum) REFERENCES COURSE(catnum)
        ON DELETE CASCADE
        ON UPDATE CASCADE
)   CHARACTER SET 'utf8mb4'
    COLLATE 'utf8mb4_unicode_520_ci';

CREATE TABLE ENROLLED (
    studentid       int          NOT NULL,   -- part of the primary key, so not null
    catnum          varchar(20)  NOT NULL,   -- part of the primary key, so not null
    sectnum         int          NOT NULL,   -- part of the primary key, so not null
    semester        varchar(20)  NOT NULL,   -- part of the primary key, so not null
    grade           varchar(2),
    rating          int,
    PRIMARY KEY (studentid, catnum, sectnum, semester),
    FOREIGN KEY (studentid) REFERENCES STUDENT(ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (catnum, sectnum, semester) REFERENCES SECTION(catnum, sectnum, semes
ter)
        ON DELETE CASCADE
        ON UPDATE CASCADE
)   CHARACTER SET 'utf8mb4'
    COLLATE 'utf8mb4_unicode_520_ci';

-- Part 2
INSERT INTO PROFESSOR(ID, first, last) VALUES (1, 'Amos', 'Burton');
INSERT INTO PROFESSOR(ID, first, last) VALUES (2, 'James', 'Holden');
INSERT INTO PROFESSOR(ID, first, last) VALUES (3,  'Prax', 'Ming');
INSERT INTO PROFESSOR(ID, first, last) VALUES (4,  'Mei', 'Ming');
INSERT INTO PROF_EMAIL(ID, email) VALUES (1, 'amos@legitimatesalvage.com');
INSERT INTO STUDENT(ID, first, last, major) VALUES (1, 'Naomi', 'Nagata', 'Physics');
INSERT INTO STUDENT(ID, first, last, major) VALUES (2, 'Chrisjen', 'Avasarala', 'Ling
uistics');
INSERT INTO STUDENT_EMAIL(ID, email) VALUES (1, 'naomi@opa.org');
INSERT INTO COURSE(catnum, description, ge_area, units)
    VALUES ('CSC134', 'Database Management Systems', NULL, 3);
INSERT INTO COURSE(catnum, description, ge_area, units)
    VALUES ('CSC174', 'Advanced Database Management Systems', NULL, 3);
INSERT INTO COURSE(catnum, description, ge_area, units)
    VALUES ('CSC130', 'Awesome Algorithms', 'G1', 2);
INSERT INTO SECTION(catnum, sectnum, semester, room_num, prof_ID)
    VALUES ('CSC174', 1, 'Spring 2020', 'RVR1002', 1);
```

```
INSERT INTO SECTION(catnum, sectnum, semester, room_num, prof_ID)
    VALUES ('CSC174', 2, 'Spring 2020', 'RVR1008', 2);
INSERT INTO SECTION(catnum, sectnum, semester, room_num, prof_ID)
    VALUES ('CSC174', 2, 'Fall 2020', 'RVR1008', 2);
INSERT INTO SECTION(catnum, sectnum, semester, room_num, prof_ID)
    VALUES ('CSC134', 5, 'Spring 2020', 'RVR2008', 1);
INSERT INTO SECTION(catnum, sectnum, semester, room_num, prof_ID)
    VALUES ('CSC130', 1, 'Spring 2019', 'RVR1002', 3);
INSERT INTO SECTION(catnum, sectnum, semester, room_num, prof_ID)
    VALUES ('CSC130', 2, 'Spring 2019', 'RVR1002', 4);
INSERT INTO ENROLLED(studentid, catnum, sectnum, semester, grade, rating)
    VALUES (1, 'CSC134', 5, 'Spring 2020', 'A', 4);
INSERT INTO ENROLLED(studentid, catnum, sectnum, semester, grade, rating)
    VALUES (2, 'CSC134', 5, 'Spring 2020', 'A', 5);
INSERT INTO ENROLLED(studentid, catnum, sectnum, semester, grade, rating)
    VALUES (1, 'CSC174', 1, 'Spring 2020', 'A', 3);
INSERT INTO ENROLLED(studentid, catnum, sectnum, semester, grade, rating)
    VALUES (2, 'CSC130', 1, 'Spring 2019', 'A', 5);

-- Part 3

-- 1. List the catalog number and description for every course that has had a section
taught in room RVR1002.
-- We need COURSE for the description and SECTION for the room number. Catnum can com
e from either of those.
-- Luckily NATURAL JOIN will work for this.
SELECT catnum, description
FROM COURSE natural join SECTION
WHERE room_num = 'RVR1002';

-- 2.   List the ID, first, and last names of all professors who taught any sections
 in Spring 2020, along with the
-- catalog number, section number, and description of what they taught that semester.
-- We'll need PROFESSOR, COURSE, and SECTION. Since we don't want a duplicate for the
catalog number,
-- we'll either need to use a NATURAL JOIN or do some ALIASING. And finally we'll sor
t the output using ORDER BY
SELECT ID, first, last, catnum, sectnum, description
FROM COURSE NATURAL JOIN SECTION JOIN PROFESSOR ON prof_ID = ID
WHERE semester = 'Spring 2020'
ORDER BY catnum, sectnum ASC;

-- 3. For each student, list their ID, first name, last name, and average rating left
by that student.
-- We need to join STUDENT and ENROLLED to find students and their ratings. Then, in
 order to use an
-- aggregate function like avg(), we need to use GROUP BY.
SELECT ID, first, last, avg(rating) as 'average rating'
FROM STUDENT JOIN ENROLLED on ID = studentid
GROUP BY ID;

-- 4. For each professor who has at least two ratings, list their ID, first name, las
t name, and the average rating left by
students for sections
```

```
-- that professor has taught.
-- This will take a few different tables joined together. We obviously need PROFESSOR
and SECTION. We'll also
-- need ENROLLED to get ratings. However, we don't actually need STUDENT, since ENROL
LED has everything we need
-- about ratings.
SELECT ID, first, last, avg(rating) as 'average rating'
FROM PROFESSOR JOIN SECTION ON ID = prof_ID NATURAL JOIN ENROLLED
GROUP BY ID
HAVING count(rating) >= 2;


-- 5. List the ID, first, and last names of all students who do NOT have an email add
ress.
-- We could take a set difference approach here, but not all implementations support
 this. Instead,
-- we can use a LEFT OUTER JOIN and look for email addresses of NULL. Since NULL ca
n't occur in the
-- actual email address, we know any NULL values from the LEFT OUTER JOIN came from s
tudents who did
-- not have an email. Remember when looking for NULL we need to use 'IS NULL', not '=
NULL'.
-- Remember that we have ID twice, so you need to disambiguate the SELECT statement.

SELECT STUDENT.ID, first, last
FROM STUDENT LEFT OUTER JOIN STUDENT_EMAIL on STUDENT.ID = STUDENT_EMAIL.ID
WHERE email is NULL;

-- 6. List the ID, first and last names of all professors who have NOT taught a secti
on taken by a student
-- majoring in physics.
-- Nested queries are really convenient for this. First we can find the ID of all pro
fessors who have taught
-- a section for such a student. That becomes the inner query, the outer query finds
 professors not in that
-- group.
-- We should alias ID in the inner query because it has access to the outer query, wh
ich also contains ID.
SELECT ID, first, last
FROM PROFESSOR
WHERE ID NOT IN (
    SELECT prof_ID
    FROM SECTION NATURAL JOIN ENROLLED JOIN STUDENT ON studentid = STUDENT.ID
    WHERE major = 'physics'
    );

-- Part 4
DROP TABLE IF EXISTS ENROLLED;
DROP TABLE IF EXISTS STUDENT_EMAIL;
DROP TABLE IF EXISTS PROF_EMAIL;
DROP TABLE IF EXISTS SECTION;
DROP TABLE IF EXISTS COURSE;
DROP TABLE IF EXISTS STUDENT;
DROP TABLE IF EXISTS PROFESSOR;
```