Theodore Kim
JinZhao Su
CS-UY 3083: Introduction to Databases
HW #4

--1.1

```
SELECT DISTINCT id, name FROM student NATURAL JOIN takes WHERE
takes.grade = "A" AND (takes.course_id = "CS-101" OR takes.course_id =
"CS-319");
```

```
--1.2

SELECT DISTINCT id, name FROM student NATURAL JOIN takes WHERE
(takes.grade = "A" AND takes.course_id = "CS-319") OR (takes.grade !=
"A" AND takes.course_id = "CS-101");
```

```
--1.3
SELECT id FROM takes GROUP BY course_id, id HAVING COUNT(*) > 1;
```

```
--1.4
CREATE TABLE gradepoint (
      letter varchar(2),
      points numeric(2, 1),
      PRIMARY KEY (letter, points)
);
INSERT INTO gradepoint VALUES ("A", 4.0), ("A-", 3.7), ("B+", 3.3),
("B", 3.0), ("B-", 2.7), ("C+", 2.3), ("C", 2.0), ("C-", 1.7), ("D+",
1.3), ("D", 1.0), ("F", 0);


CREATE VIEW StudentGPA AS
SELECT t1.id, (SUM(weighted_points) / total_credits) AS gpa
FROM (SELECT takes.id,
      (course.credits * gradepoint.points) AS weighted_points
      FROM takes
      NATURAL JOIN course
      JOIN gradepoint ON gradepoint.letter = takes.grade) AS t1
NATURAL JOIN (SELECT takes.id, SUM(course.credits) AS total_credits
              FROM takes NATURAL JOIN course GROUP BY takes.id) AS t2
GROUP BY id;
```

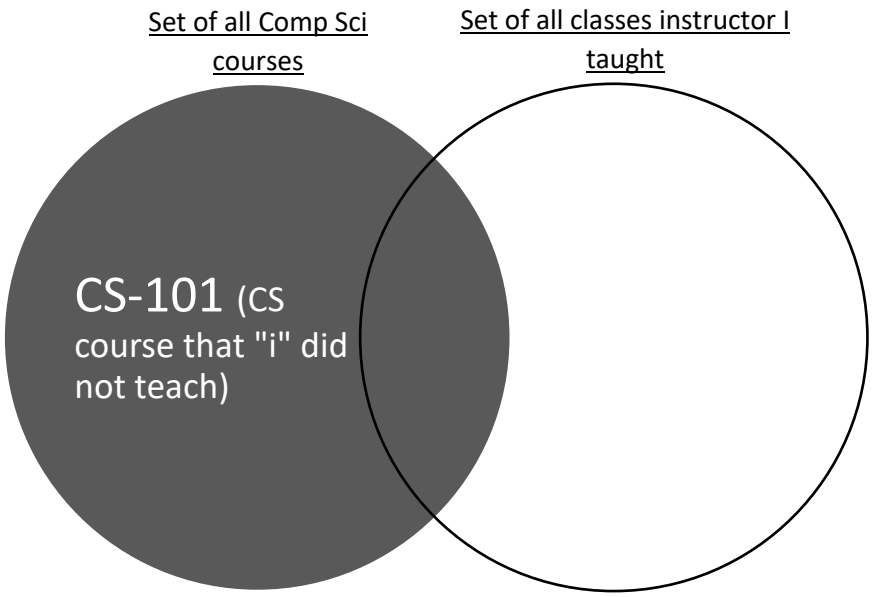
```
--1.5

--The foreign constraint belongs on the takes table to make sure that
the grade column corresponds to an actual letter grade


ALTER TABLE takes
ADD CONSTRAINT FOREIGN KEY (grade) REFERENCES gradepoint.letter;
```

```
--1.6

SELECT DISTINCT takes.id
FROM takes
JOIN gradepoint ON letter=grade
JOIN (SELECT id, MAX(gradepoint.points) AS maxgrade, course_id FROM
takes JOIN gradepoint ON letter=grade WHERE course_id="CS-101" OR
course_id="CS-347" GROUP BY id HAVING course_id="CS-347") AS t1 ON
t1.id = takes.id AND maxgrade = gradepoint.points
WHERE takes.course_id="CS-101" OR takes.course_id="CS-347"
```

```
--1.7

SELECT DISTINCT id
FROM takes
WHERE (grade = "A" OR grade = "A-") AND id NOT IN (SELECT DISTINCT id
FROM takes WHERE grade != "A" AND grade != "A-" GROUP BY id);
```

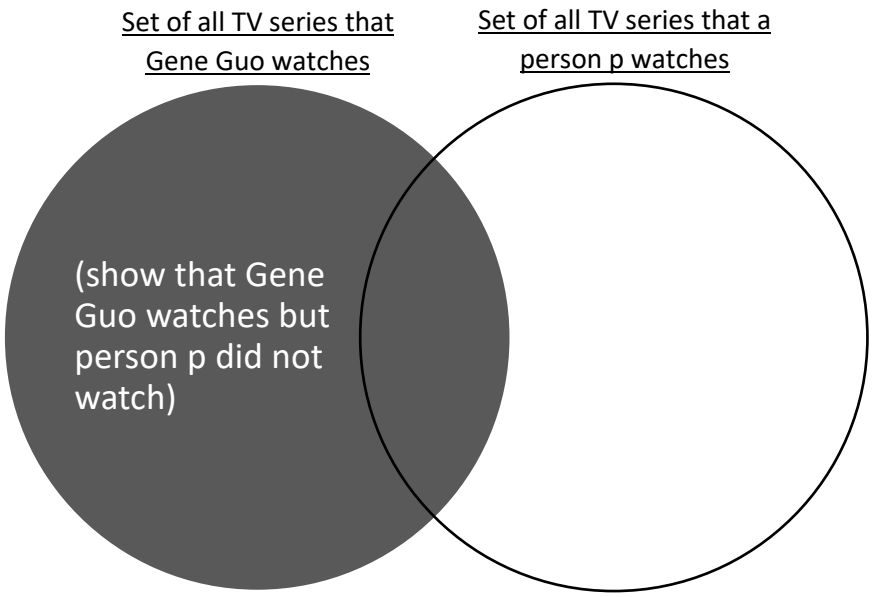
```
--1.8

SELECT section.course_id
FROM section
INNER JOIN (SELECT course_id, year FROM section) AS t1
ON t1.year = section.year + 1 AND section.course_id = t1.course_id
```

--1.9




```
--Write a query to find the number of Comp Sci courses that instructor 12345 taught.

SELECT COUNT(DISTINCT course_id) FROM teaches WHERE ID=12345 AND course_id LIKE "CS%"

--Write a query to find the total number of Comp Sci courses

SELECT COUNT(course_id) FROM course WHERE dept_name="Comp. Sci."

--Write a predicate that uses the above two counting queries and that is true if and only if instructor 12345 taught all Comp Sci courses.

(SELECT COUNT(DISTINCT course_id) FROM teaches WHERE ID=12345 AND course_id LIKE "CS%") = (SELECT COUNT(course_id) FROM course WHERE dept_name="Comp. Sci.")

--Now, instead of focusing on instructor 12345, enclose the predicate you just wrote in a query that will check whether an arbitrary instructor i taught all Comp Sci courses.

SELECT ID FROM instructor NATURAL JOIN teaches WHERE teaches.course_id LIKE "CS%" GROUP BY ID HAVING COUNT(*) = (SELECT COUNT(*) FROM course WHERE dept_name="Comp. Sci.");
```

--1.10




```
--Number of shows that Gene Guo watches
SELECT COUNT(*) FROM watches WHERE first_name = "Gene" AND last_name =
"Guo"

--Number of shows a person watches
SELECT COUNT(*) FROM watches GROUP BY first_name, last_name

--Predicate
SELECT COUNT(*) FROM watches GROUP BY first_name, last_name HAVING
COUNT(*) = (SELECT COUNT(*) FROM watches WHERE first_name = "Gene" AND
last_name = "Guo")

--Final Query
SELECT first_name, last_name
FROM watches
JOIN (SELECT DISTINCT title, year FROM watches WHERE first_name =
"Gene" AND last_name = "Guo") AS t1 ON t1.title=watches.title
GROUP BY first_name, last_name
HAVING first_name != "Gene"
      AND last_name != "Guo"
      AND COUNT(*) = (SELECT COUNT(*) FROM watches WHERE first_name =
            "Gene" AND last_name = "Guo");
```

```
--1.11

SELECT tvseries.title
FROM tvseries
LEFT JOIN watches ON watches.title = tvseries.title
WHERE watches.first_name IS NULL
```

```sql
--1.12

SELECT course.course_id, ID
FROM course
LEFT JOIN teaches ON course.course_id = teaches.course_id
```

--2.1

If an instructor is changed, then all tables that rely on the instructor table as a foreign key will also be altered. These tables are: **teaches** and **advisor.**

The table teaches will have rows whose columns refer to the deleted row in instructor **deleted** as the foreign key constraint on that table has set "ON DELETE CASCADE" specified.

The table advisor will have the value of i_ID set to NULL when the corresponding row in instructor is deleted.

--2.2
If the ON DELETE SET NULL was replaced with ON DELETE CASCADE, then the foreign key constraint on the advisor table would change. Now, if an instructor is deleted, then the corresponding row of the advisor table would also be deleted.

```
--2.3

CREATE FUNCTION untaught_section (id int, cid varchar(8), s
varchar(6), y decimal(4, 0))
RETURNS integer
BEGIN
      DECLARE s_count INTEGER;
      SELECT COUNT(*) INTO s_count
      FROM section
      LEFT JOIN teaches ON teaches.sec_id = section.sec_id AND
      teaches.course_id = section.course_id
      WHERE section.sec_id = id
      AND teaches.ID IS NOT NULL
      AND section.course_id = cid
      AND section.semester = s
      AND section.year = y;
      RETURN s_count;
END

ALTER TABLE takes
ADD CONSTRAINT untaught_constraint CHECK (untaught_section(sec_id,
course_id, semester, year) > 0)
```