# ASSIGNMENT#4

by Karla Martinez

## EXPLAIN THE DIFFERENCE BETWEEN A WEAK AND A STRONG ENTITY SET. USE AN EXAMPLE OTHER THAN THE ONE IN CHAPTER 6 TO ILLUSTRATE. (CONSULT CH. 6, 6.5.3)

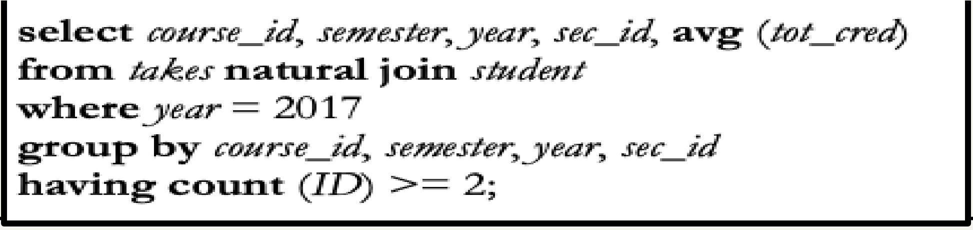
- An entity set that has a primary key is termed a strong entity set.

- A **weak entity set** is one whose existence is dependent on another entity set, called its **identifying entity set**; instead of associating a primary key with a weak entity, we use the primary key of the identifying entity, along with extra attributes, called **discriminator attributes** to uniquely identify a weak entity.
  - A song depends on the album or artist for its existence
- A strong entity is independent
  - EX: Artist can be independence and has its own uni identifyer

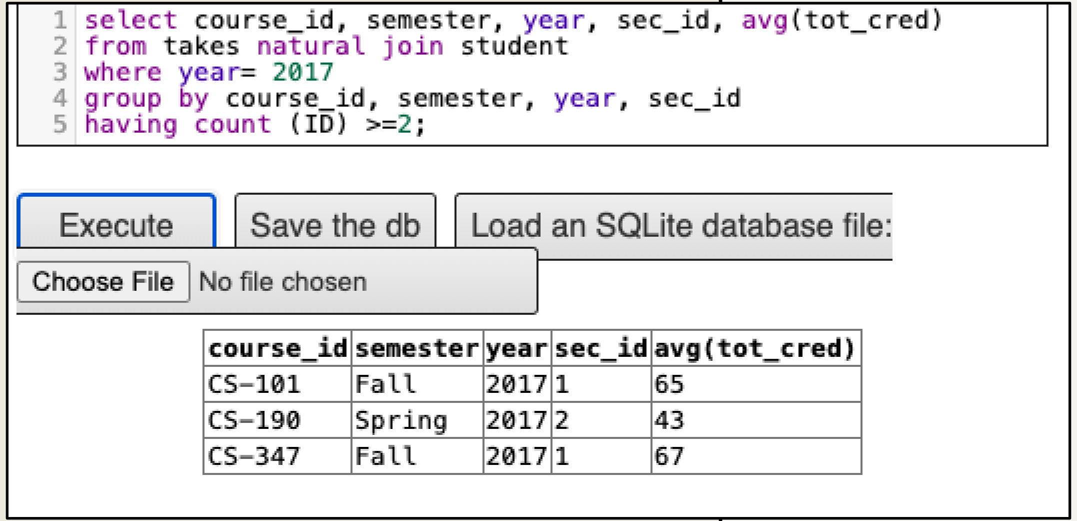
# DESIGN AN E-R DIAGRAM FOR KEEPING TRACK OF THE SCORING STATISTICS OF YOUR FAVORITE SPORTS TEAM. YOU SHOULD STORE THE MATCHES PLAYED, THE SCORES IN EACH MATCH, THE PLAYERS IN EACH MATCH, AND INDIVIDUAL PLAYER SCORING STATISTICS FOR EACH MATCH. SUMMARY STATISTICS SHOULD BE MODELED AS DERIVED ATTRIBUTES WITH AN EXPLANATION AS TO HOW THEY ARE COMPUTED.

- Draw the E-R diagram using draw.io. Read this website for instructions.
- b) Expand to all teams in the league (Hint: add team entity)

# CONSIDER THE QUERY

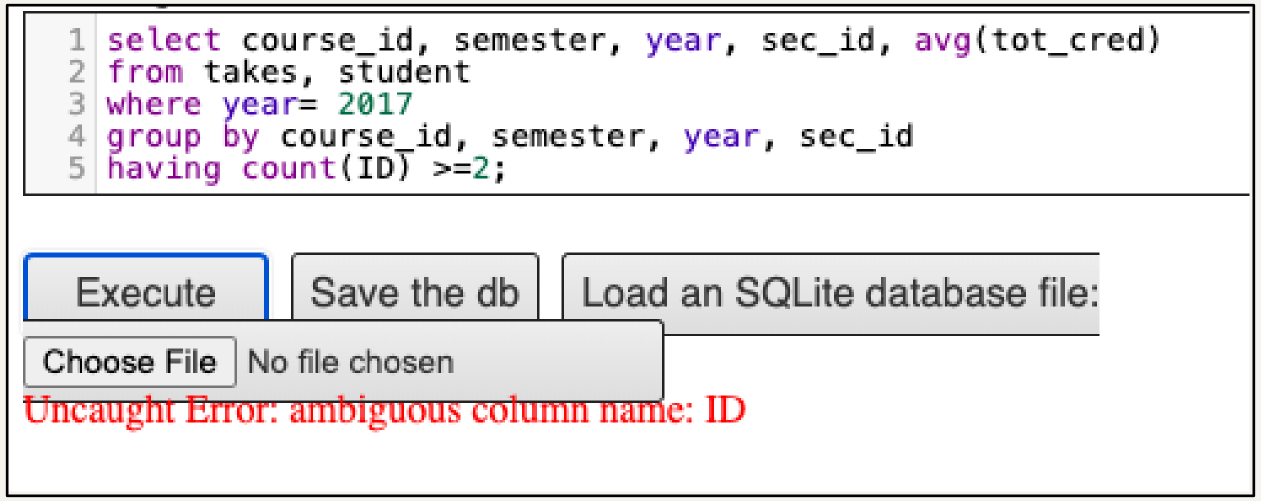


**select** *course_id*, *semester*, *year*, *sec_id*, **avg** (*tot_cred*)
**from** *takes* **natural join** *student*
**where** *year* = 2017
**group by** *course_id*, *semester*, *year*, *sec_id*
**having count** (*ID*) >= 2;



```
select course_id, semester, year, sec_id, avg(tot_cred)
from takes natural join student
where year= 2017
group by course_id, semester, year, sec_id
having count (ID) >=2;
```

| course_id | semester | year | sec_id | avg(tot_cred) |
|---|---|---|---|---|
| CS-101 | Fall | 2017 | 1 | 65 |
| CS-190 | Spring | 2017 | 2 | 43 |
| CS-347 | Fall | 2017 | 1 | 67 |



```
select course_id, semester, year, sec_id, avg(tot_cred)
from takes, student
where year= 2017
group by course_id, semester, year, sec_id
having count(ID) >=2;
```

Uncaught Error: ambiguous column name: ID

- Explain why appending **natural join** *section* in the **from** clause would not change the result. (Consult Ch. 4, 4.1.1)

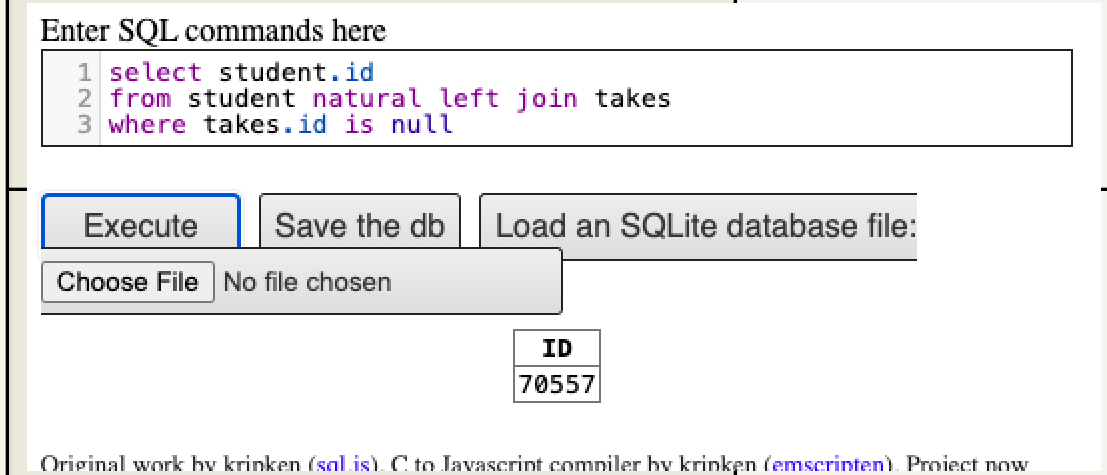
- Test the results using the Online SQL interpreter (https://www.db-book.com/db7/university-lab-dir/sqljs.html)

The appending natural join will not change results because it is a simpler way for an SQL programmer to show information from two or more relations joined together.. It operated on two relationships and produces a relation . Therefore considering this query, both the tuple from takes and the tuples from students have the same value on common attribute . Would basically be the same as stating from takes, student.

# CONSIDER THE QUERY

Write an SQL query using the university schema to find the ID of each student who has never taken a course at the university. Do this using no subqueries and no set operations (use an outer join). (Consult Ch. 4, 4.1.3)

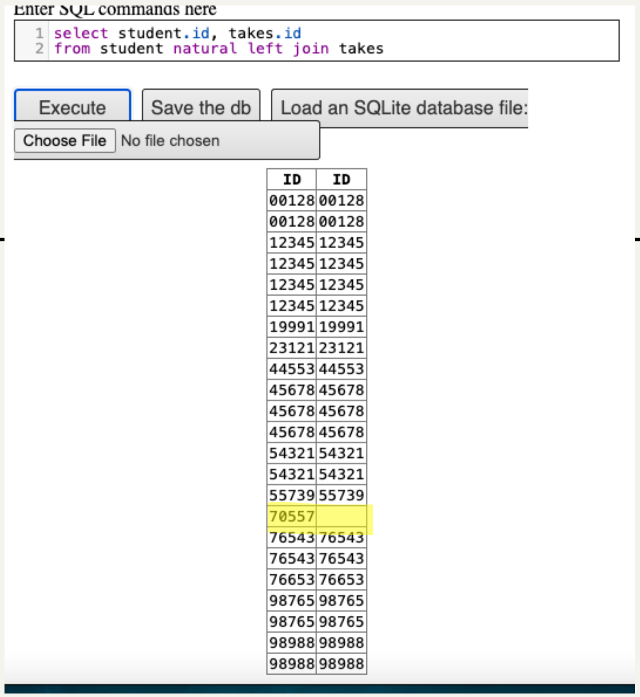
Per the book "outer-join operation works in a manner similar to the join operations we have already studied, but it preserves those tuples that would be lost in a join by creating tuples in the result containing null values."



```
select student.id
from student natural left join takes
where takes.id is null
```

| ID |
|---|
| 70557 |

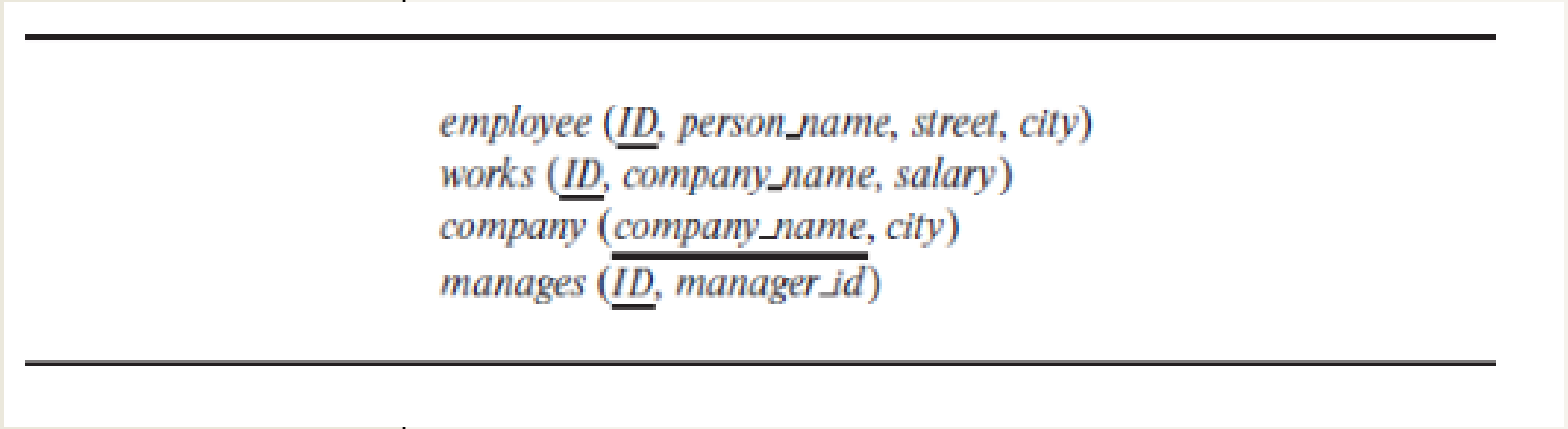
Original work by kripken (sql.js). C to Javascript compiler by kripken (emscripten). Project now



```
select student.id, takes.id
from student natural left join takes
```

| ID | ID |
|---|---|
| 00128 | 00128 |
| 00128 | 00128 |
| 12345 | 12345 |
| 12345 | 12345 |
| 12345 | 12345 |
| 12345 | 12345 |
| 19991 | 19991 |
| 23121 | 23121 |
| 44553 | 44553 |
| 45678 | 45678 |
| 45678 | 45678 |
| 45678 | 45678 |
| 54321 | 54321 |
| 54321 | 54321 |
| 55739 | 55739 |
| 70557 | |
| 76543 | 76543 |
| 76543 | 76543 |
| 76653 | 76653 |
| 98765 | 98765 |
| 98765 | 98765 |
| 98988 | 98988 |
| 98988 | 98988 |

## CONSIDER THE FOLLOWING DATABASE, WRITE A QUERY TO FIND THE ID OF EACH EMPLOYEE WITH NO MANAGER. NOTE THAT AN EMPLOYEE MAY SIMPLY HAVE NO MANAGER LISTED OR MAY HAVE A *NULL* MANAGER(USE NATURAL LEFT OUTER JOIN). (CONSULT CH. 4, 4.1.3)



*employee* (*ID*, *person_name*, *street*, *city*)
*works* (*ID*, *company_name*, *salary*)
*company* (*company_name*, *city*)
*manages* (*ID*, *manager_id*)

```
Select employee.id
From employee natural left outer join manages
Where manages.id is null

Select employee.id
From employee natural left outer join manager
```