1. **Visualizing Joins:**

Available tables:

student table:

student_ID	first_name	last_name	main_campus	advisor
1	~	~	~	~
2	~	~	~	~
3	~	~	~	~
4	~	~	~	~

course table: [simplified model to maintain 2 tables only, we'll review joining more tables in the next class]

course_ID	course_name	classroom	student_ID
1	~	~	1
2	~	~	1
3	~	~	2
4	~	~	3

(most, would be in a class, but some would not. What do we do with them?)

1) **Start with a simple query**:

```
SELECT *    <-Select all columns from all tables of interests
FROM student <-main table we want to get data from, also known as "Left Table"
INNER JOIN course <- table we want to get additional data on and connect it to the Left table, also known as "Right Table"
ON student.student_ID = course.student_ID <- how we want to connect the rows in these tables
; <- finishes the query (we can write another query right after this semicolon symbol)
```

2) **Alternatively**:

```
SELECT *
FROM student
WHERE student.student_ID = course.student_ID <- alternatively, we can also use a where clause to identify how we want to join two tables
;
```

3) **Now let's refine it**:

```
SELECT first_name, last_name  <- choose variables of interest
FROM student
INNER JOIN course
ON student.student_ID = course.student_ID
;
```

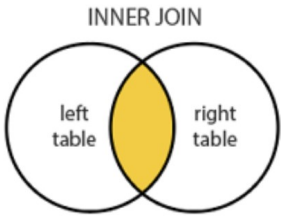
4) Standard notation:

```
SELECT colum1, column2, …
FROM [table_of_interest]
_____ [type_of_Join] JOIN [table_we_want_to_connect_with]
ON firsttable.[variable_used_to_connect_in_firsttable] = secondtable.
[variable_used_to_connect_in_secondtable]
WHERE condition(s)
ORDER BY vale_to_order_by
;
```

5) Types of Join:
    a. **Inner Join**: Only returns connecting matching rows.

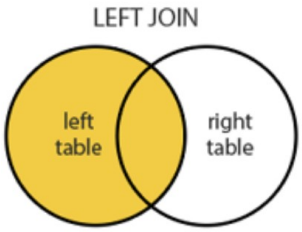
```
SELECT student.first_name, student.last_name, course.course_name, course.course_ID
FROM student
INNER JOIN course
ON student.student_ID = course.student_ID
;
```

In our previous example, we are picking information FROM the left table, which is the student table, and matching it to the right table which becomes the course table. This is also valid for the examples below.



This will only give us all students taking a course with the corresponding course information.

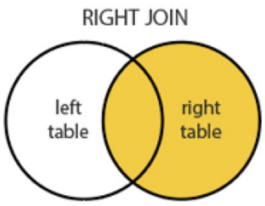
b. **Left Join**: Return all connected rows + all unconnected rows from Left table (nulls in missing values).




```
SELECT student.first_name, student.last_name, course.course_name, course.course_ID
FROM student
LEFT JOIN course
ON student.student_ID = course.student_ID
;
```

This will give us all students, regardless of whether they are currently taking a course, AND all students taking a course with the corresponding course information.

c. **Right Join**: Return all connected rows + all unconnected rows from Right table (nulls in missing values).

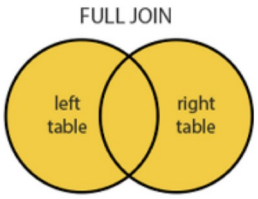



```
SELECT student.first_name, student.last_name, course.course_name, course.course_ID
FROM student
RIGHT JOIN course
```

```
ON student.student_ID = course.student_ID
;
```

This will give us all courses, regardless of whether they have students currently taking them, AND all courses with students in them and the corresponding information about those students.

d. **Full Join**: Return all connected rows + all unconnected rows from Both tables (nulls in missing values).




```
SELECT student.first_name, student.last_name, course.course_name, course.course_ID
FROM student
FULL JOIN course
ON student.student_ID = course.student_ID
;
```

6) **Subqueries**:

We can also use the results of a query as the basis for another query. To exemplify this, we 1) first need to create a query to identify the data we are interested in, 2) we need to provide an alias for the results so we can refer to them in the same way we normally refer to a table, and 3) we need to write the query that will use this results.

Let us start with identifying the courses that student ID 1 has been taking (we will use this new table to match it with the student's table and get more data on student 1).

```
SELECT *
FROM course
WHERE course.student_ID = 1
;
```

This query will give us a table with all the course table's attributes but only those rows with student ID 1.

course_ID	course_name	classroom	student_ID
1	~	~	1

2	~	~	1

Now we can give this new table an alias so that we can refer to it in other parts of the query:

```
(SELECT *
FROM course
WHERE course.student_ID = 1)
;
```

Or

```
(SELECT * FROM course WHERE course.student_ID = 1) AS c
;
```

Finally, we can write another query that will use this table and match it with a different table (namely the student table) and join them so that we get all of the data for student 1 attached to the table above.

```
SELECT *
FROM (SELECT * FROM course WHERE course.student_ID = 1) AS c <- we rename this table as c so we just need to write "c" to refer to it
RIGHT JOIN student AS s <- we rename the student table as s so we just need to write "s" to refer to it
ON c.student_ID = s.student_ID <- we match the student_ID in the newly created table, with the student ID in the student table
ORDER BY c.course_ID;
;
```

7) **Self Join**:

```
SELECT *
FROM student As s
INNER JOIN course AS c
ON s.student_ID = c.student_ID
ORDER BY s.student_ID
;
```

8) **Identify those with Null values (IS NULL)**:

```
SELECT *
FROM student As s
FULL JOIN course AS c
ON s.student_ID = c.student_ID
```

```
WHERE s.student_ID IS NULL OR c.student_ID IS NULL
;
```

The table showing from this query will have only results with a NULL student ID in either the student table or the course table.