
Agenda

- Joins
- Self Join
    - SQL query as pseudocode
- Joining Multiple Tables

Joins

Today we are going to up the complexity of SQL Read queries we are going to write while still using the same foundational concepts we had learnt in the previous class on CRUD. Till now, whenever we had written an SQL query, the query found data from how many tables?

Correct, every SQL query we had written till now was only finding data from 1 table. Most of the queries we had written in the previous class were on the `film` table where we applied multiple filters etc. But do you think being able to query data from a single table is enough? Let's take a scenario of Scaler. Let's say we have 2 tables as follows in the Scaler's database:

`batches`

batch_id	batch_name
1	Batch A
2	Batch B
3	Batch C

`students`

student_id	first_name	last_name	batch_id
1	John	Doe	1
2	Jane	Doe	1
3	Jim	Brown	2
4	Jenny	Smith	3
5	Jack	Johnson	2

Suppose, someone asks you to print the name of every student, along with the name of their batch. The output should be something like:

student_name	batch_name
John	Batch A
Jane	Batch A
Jim	Batch B
Jenny	Batch C

Jack	Batch B

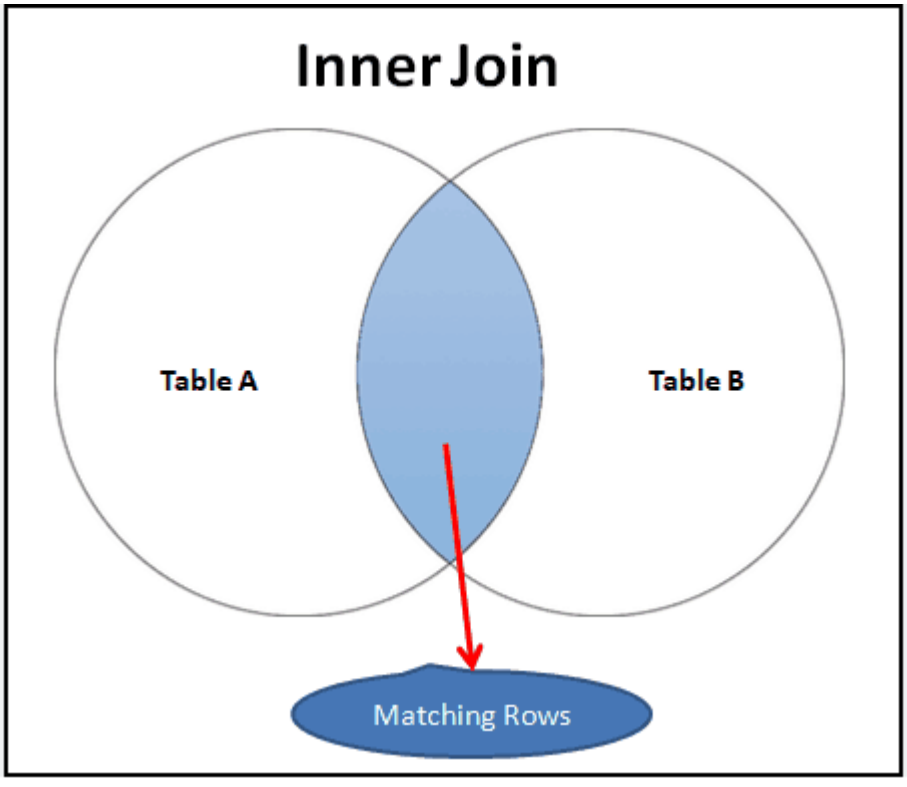
Will you be able to get all of this data by querying over a single table? No. The `student_name` is there in the students table, while the `batch_name` is in the batches table! We somehow need a way to combine the data from both the tables. This is where joins come in. What does the word `join` mean to you?

Correct! Joins, as the name suggests, are a way to combine data from multiple tables. For example, if we want to combine the data from the `students` and `batches` table, we can use joins for that. Think of joins as a way to stitch rows of 2 tables together, based on the condition you specify. Example: In our case, we would want to stitch a row of students table with a row of batches table based on what? Imagine that every row of `students` we try to match with every row of `batches`. Based on what condition to be true between those will we stitch them?

Correct, we would want to stitch a row of students table with a row of batches table based on the `batch_id` column. This is what we call a `join condition`. A join condition is a condition that must be true between the rows of 2 tables for them to be stitched together.

Let's try to understand this with a Venn diagram:

> *Venn Diagram:*



> *Source: Unknown*

Let's see how we can write a join query for our example.

```sql
SELECT students.first_name, batches.batch_name
FROM students
JOIN batches
ON students.batch_id = batches.batch_id;
```

Let's break down this query. The first line is the same as what we have been writing till now. We are selecting the `first_name` column from the `students` table and the `batch_name` column from the `batches` table. The next line is where the magic happens. We are using the `JOIN` keyword to tell SQL that we want to join the `students` table with the `batches` table. The next line is the join condition. We are saying that we want to join the rows of `students` table with the rows of `batches` table where the `batch_id` column of `students` table is equal to the `batch_id` column of `batches` table. This is how we write a join query.

Let's take an example of this on the Sakila database. Let's say for every film, we want to print its name and the language. How can we do that?

```sql
SELECT film.title, language.name
FROM film
JOIN language
ON film.language_id = language.language_id;
```

Now, sometimes typing name of tables in the query can become difficult. For example, in the above query, we have to type `film` and `language` multiple times. To make this easier, we can give aliases to the tables. For example, we can give the alias `f` to the `film` table and `l` to the `language` table. We can then use these aliases in our query. Let's see how we can do that:

```sql
SELECT f.title, l.name
FROM film f
JOIN language l
ON f.language_id = l.language_id;

-- These aliases are even more helpful in self joins
```

This above join is also known as Inner Join. We will talk more about Inner and Outer joins in next topic's notes.

If you want to know more about this topic you may visit:
https://scaler.com/topics/inner-join-in-sql/

Visual Description using one more table example:

We will use example of “Students” table and a “Batch” table again.

> *Students Table:*

student id	name	name
1	James Jones	1
2	John Miller	2
3	John Martinez	3
4	Michael Garcia	4
5	Patricia Martin	1
6	Michael Rodrig	2
7	Robert Brown	3
8	Patricia Brown	4
9	Patricia William	1
10	Jennifer Miller	2

Batches Table:

batch id	batch_name
1	Batch A
2	Batch B
3	Batch C
4	Batch D

Lets use the SQL query again:

```
SELECT students.first_name, batches.batch_name
FROM students
```

```
JOIN batches
ON students.batch_id = batches.batch_id;
```

Here for this query each value in **Student's batch_id** column is matched with each value in **Batches's batch_id** column as described in following pseudo code.

In pseudocode, it shall look like:

```
ans = []

for row1 in students:
    for row2 in batches:
        if row1.batch_id == row2.batch_id:
            ans.add(row1 + row2)

for row in ans:
    print(row.name, row.name)
```

Now, the final table will look like following one where light blue column belongs to Student's table and magenta color columns belong to Batches table in this resultant table:

> *Resultant Table:*

student id	name	batch id	batch	batch_name
1	James Jones	1	1	Batch A
2	John Miller	2	2	Batch B
3	John Martinez	3	3	Batch C
4	Michael Garcia	4	4	Batch D
5	Patricia Martine:	1	1	Batch A
6	Michael Rodrigu	2	2	Batch B
7	Robert Brown	3	3	Batch C
8	Patricia Brown	4	4	Batch D
9	Patricia Williams	1	1	Batch A
10	Jennifer Miller	2	2	Batch B

Now from this table we can print any columns using the table name aliases. For example if we want to print student's name and batches name then we may write following inside select command:

```
SELECT students.first_name, batches.batch_name
```

> *Activity: Try `Select *` for the above query.*

Self Join

Let's say at Scaler, for every student we assign a Buddy. For this we have a `students` table, which has following columns/fields:

`id | name | buddy_id`

This `buddy_id` will be an id of what?

> *NOTE: Give hints to get someone to say `student`*

Correct. Now, let's say we have to print for every student, their name and their buddy's name. How will we do that? Here 2 rows of which tables would we want to stitch together to get this data?

Correct, an SQL query for the same shall look like:

```
SELECT s1.name, s2.name
FROM students s1
JOIN students s2
ON s1.buddy_id = s2.id;
```

This is an example of SELF join. A self join is a join where we are joining a table with itself. In the above query, we are joining the `students` table with itself. In a self joining, aliasing tables is very important. If we don't alias the tables, then SQL will not know which row of the table to match with which row of the same table (because both of them have same names as they are the same table only). Please refer to following picture.

> *Note: Do remember that in self join too the matching row for given conditions will be present in the output/resultant table.*

> *Venn Diagram:*

> *Source: Unknown*

Please try this above query once by yourself.

Consider following infographics to understand above query:

In this table, each student is assigned a 'Buddy', now we have to find buddies of every student.

id	name	buddy_id
1	Sophia Martinez	10
2	Elijah Brown	9
3	Ava Davis	8
4	Noah Martinez	7
5	Noah Jones	6
6	Elijah Davis	5
7	Sophia Garcia	4
8	Charlotte Williams	3
9	Liam Garcia	2
10	Oliver Williams	1

To find each student’s buddy, we used a self-join to stitch together two rows of our table. Let's see how this works in practice.

```
SELECT s1.name, s2.name
FROM students t1
JOIN students t2
ON s1.buddy_id = s2.id;
```

After combining above table we will get following output:

t1.id	t1.name	t1.buddy_id	t2.id	t2.name
1	Sophia Martinez	10	10	Oliver W
2	Elijah Brown	9	9	Liam Ga
3	Ava Davis	8	8	Charlotte
4	Noah Martinez	7	7	Sophia o
5	Noah Jones	6	6	Elijah Da
6	Elijah Davis	5	5	Noah Jo
7	Sophia Garcia	4	4	Noah Ma
8	Charlotte Williams	3	3	Ava Dav
9	Liam Garcia	2	2	Elijah Br
10	Oliver Williams	1	1	Sophia N

Now that we have final table let's print t1.name and t2.name i.e name of student and their buddy i.e final answer:

t1.name	t2.name
Sophia Martinez	Oliver W
Elijah Brown	Liam Ga
Ava Davis	Charlotte
Noah Martinez	Sophia o
Noah Jones	Elijah Da
Elijah Davis	Noah Jo
Sophia Garcia	Noah Ma
Charlotte Williams	Ava Dav
Liam Garcia	Elijah Br
Oliver Williams	Sophia N

SQL query as pseudocode (Self Join)

As we have been doing since CRUD queries, let's also see how Joins can be represented in terms of pseudocode.

Let's take this query:

```
SELECT s1.name, s2.name
FROM students s1
JOIN students s2
ON s1.buddy_id = s2.id;
```

In pseudocode, it shall look like:

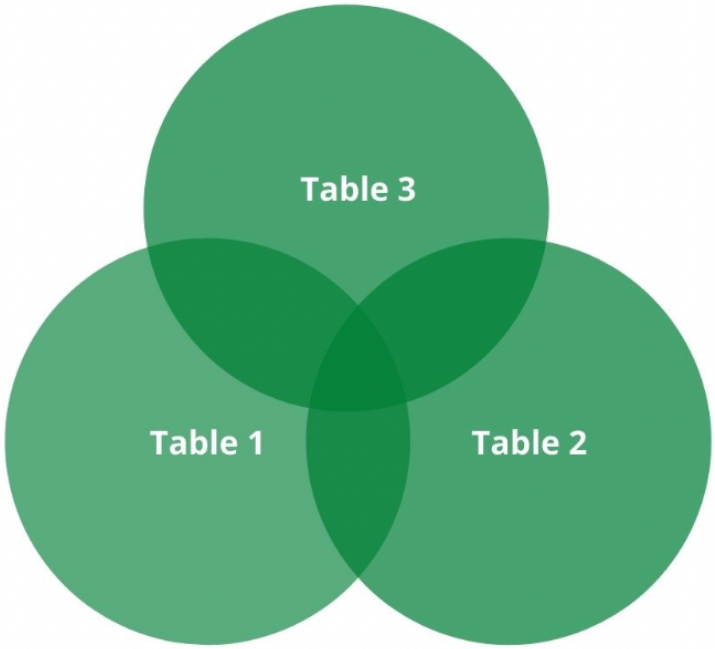
```
ans = []

for row1 in students:
    for row2 in students:
        if row1.buddy_id == row2.id:
            ans.add(row1 + row2)

for row in ans:
    print(row.name, row.name)
```

Additional resources for self joins: https://www.scaler.com/topics/sql/self-join-in-sql/

Joining Multiple Tables

Till now, we had only joined 2 tables. But what if we want to join more than 2 tables? Let's say we want to print the name of every film, along with the name of the language and the name of the original language. How can we do that? If you have to add 3 numbers, how do you do that? Correct! we add 2 numbers then add 3rd number to their sum.

To get the name of the language, we would first want to combine `film` and `language` table over the `language_id` column which will also return a table (Let's say an intermediatory table for now). Then, we would want to combine this resultant table with the language table again over the `original_language_id` column. This is how we can do that:



Source: Unknown

```
SELECT f.title, l1.name, l2.name
FROM film f
JOIN language l1
ON f.language_id = l1.language_id
JOIN language l2
ON f.original_language_id = l2.language_id;
```

Let's see how this might work in terms of pseudocode:

```
ans = []

for row1 in film:
    for row2 in language:
        if row1.language_id == row2.id:
            ans.add(row1 + row2)

for row in ans:
    for row3 in language:
        if row.language_id == row3.language_id:
            ans.add(row + row3)

for row in ans:
    print(row.name, row.language_name, row.original_language_name)
```

Activity: Please try the above query once by yourself.

Let's see how does the above query looks in execution:

`Film`

film_id	title	language_id	original_language_id
1	Space Adventure	1	2
2	Ocean's Mystery	2	1
3	Mountain Legend	1	3
4	Urban Dreams	3	2
5	Desert Mirage	2	3
6	Forest Echoes	3	1
7	Skyward Bound	1	2
8	Arctic Voyage	3	1

`Language`

language_id	name
1	English
2	Spanish
3	French
4	German
5	Italian
6	Japanese
7	Chinese
8	Russian

Expected output: Name of every film, along with the name of the language and the name of the original language.

`Output`

title	language_name	original_language_name
...	...	...
...	...	...
...	...	...
...	...	...

To get the name of the language, we would first want to combine film and language table over the language_id column:

film_id	title	language_id	original_language_id	language_name
1	Space Adventure	1	2	English
2	Ocean's Mystery	2	1	Spanis
3	Mountain Legend	1	3	English
4	Urban Dreams	3	2	French
5	Desert Mirage	2	3	Spanis
6	Forest Echoes	3	1	French
7	Skyward Bound	1	2	English
8	Arctic Voyage	3	1	French

Then, we would want to combine the result of that with the language table again over the original_language_id column.

film id	title	language id	original l_language_id	language_name	original_language_name
1	Space Adventure	1	2	English	Spanish
2	Ocean's Mystery	2	1	Spanish	English
3	Mountain Legend	1	3	English	French
4	Urban Dreams	3	2	French	Spanish
5	Desert Mirage	2	3	Spanish	French
6	Forest Echoes	3	1	French	English
7	Skyward Bound	1	2	English	Spanish
8	Arctic Voyage	3	1	French	English

Now we can easily print the highlighted tables as output: `Final Output:`

title	language_name	original_language_name
Space Adventure	English	Spanish
Ocean's Mystery	Spanish	English
Mountain Legend	English	French
Urban Dreams	French	Spanish
Desert Mirage	Spanish	French
Forest Echoes	French	English
Skyward Bound	English	Spanish
Arctic Voyage	French	English

Order of execution:

Order of Execution of a SQL query:

- **FROM** - The database gets the data from tables in FROM .
- **JOIN** - Depending on the type of JOIN used in the query and conditions specified for joining the tables in the ON clause, the database engine matches rows from the virtual table created in the FROM clause.
- **WHERE** - After the JOIN operation, the data is filtered based on the conditions specified in the WHERE clause. Rows that do not meet the criteria are excluded.
- **GROUP BY** - If the query includes a GROUP BY clause, the rows are grouped based on the specified columns and aggregate functions are applied to the groups created.
- **HAVING** - The HAVING clause filters the groups of rows based on the specified conditions
- **SELECT** - After grouping and filtering is done, the SELECT statement determines which columns to include in the final result set.
- **ORDER BY** - It allows you to sort the result set based on one or more columns, either in ascending or descending order.
- **OFFSET** - The specified number of rows are skipped from the beginning of the result set.
- **LIMIT** - After skipping the rows, the LIMIT clause is applied to restrict the number of rows returned.

> ***Note: The type of joins discussed here are also known as Inner Joins.***

Conclusion:

- Inner join in SQL selects all the rows from two or more tables with matching column values.
- Inner join can be considered as finding the intersection of two sets/Tables.

CMU notes for Joins (Too advance):
https://15445.courses.cs.cmu.edu/fall2022/slides/11-joins.pdf

Anshuman's Notes: https://docs.google.com/document/d/1TIFDVQ1Ok9ZJWTxMyJuvG5-KVwS_8_DeOnuqcDqzvbY/edit#heading=h.2s8eyo1