
Become an SQL Master
With 1 Simple Trick

Use CTEs.

Jump to page 15 if you are impatient to see the final example.

A Common Table Expression, is a temporary result set that you can reference within another SELECT, INSERT, UPDATE, or DELETE statement.

Think of it like creating **a temporary table which you can then query** as part of your overall SQL statement.

They are used to simplify complex queries, breaking them down into simpler, more readable components.

Let’s see an example!

Example #1

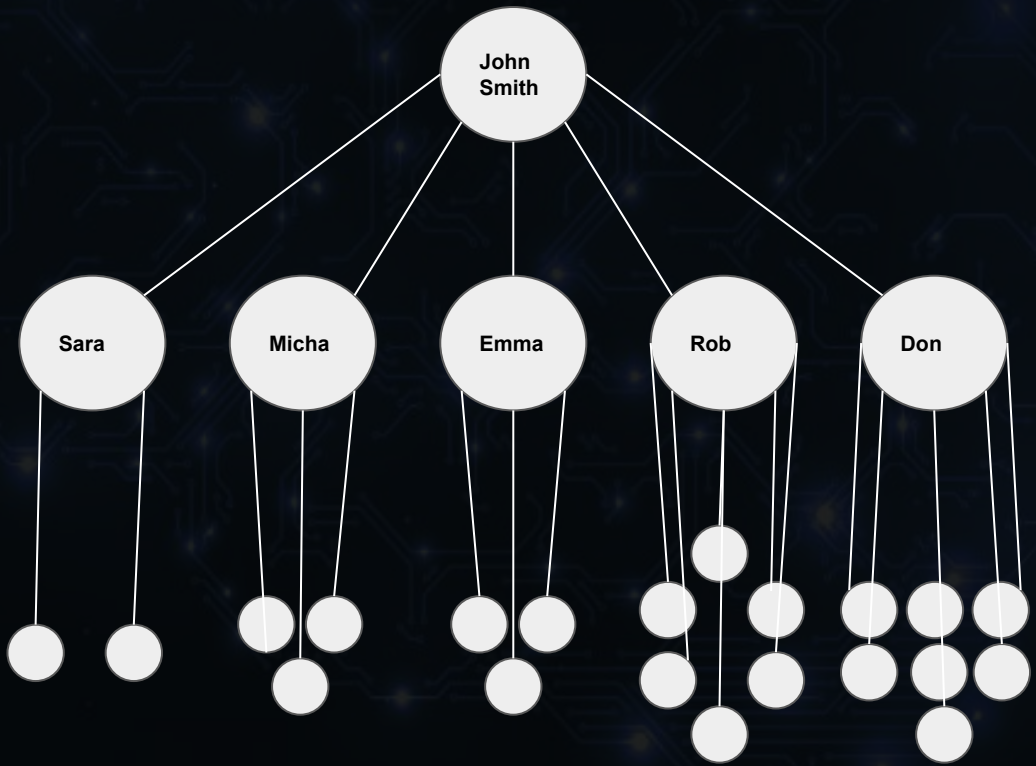
We'll use the famous 'employees' table for the example.

Each employee has an id, a name, a manager_id and a salary.

(I didn't use FKs or a sequence to simplify matters)

```
1 CREATE TABLE employees (
2     ID INT PRIMARY KEY,
3     NAME VARCHAR(100),
4     manager_id INT,
5     salary DECIMAL(10,2)
6 );
```

We'll insert data to generate the following org chart.



Select * from employees

id	name	manager_id	salary
1	John Smith	NULL	100000.00
2	Sara	1	80000.00
3	Micha	1	80000.00
4	Emma	1	80000.00
5	Rob	1	80000.00
6	Don	1	80000.00
7	Employee 7	2	60000.00
8	Employee 8	2	60000.00
9	Employee 9	3	60000.00
10	Employee 10	3	60000.00

*You can follow along using sqliteonline.com!

Let’s start.

You need to select all the employees in big teams (>5 employees in the team).

Try it out!

Option #1 - Using Subquery

```sql
SELECT e.*
FROM employees e
WHERE e.manager_id IN (
    SELECT e.manager_id
    FROM employees e
    GROUP BY e.manager_id
    HAVING COUNT(*) > 5
);
```

Option #2 - Using CTE

```sql
WITH manager_teams AS (
    SELECT manager_id, COUNT(*) AS team_size
    FROM employees
    GROUP BY manager_id
)
SELECT e.*, mt.team_size
FROM employees e
JOIN manager_teams mt ON e.manager_id = mt.manager_id
WHERE mt.team_size > 5
```

3 main CTE advantages:

1. **Readability and Maintainability**: Instead of having complex and nested subqueries, you can create a CTE at the beginning of your query. This makes the logic of your query easier to understand.

2. **Reusable** within a Single Query: Once a CTE is defined, it can be referred to multiple times within the same query.

3. **Recursive Queries**: Unlike subqueries, CTEs can be used to execute recursive queries, which are queries that refer to themselves. This is extremely useful for dealing with hierarchical or tree-structured data

Try solving this without CTE:

Find all managers of big & well paid teams:

Managers who manage more than five employees and whose teams have an average salary that is greater than the overall average salary.

Still not impressed?

We are getting to the BEST part.

For the final example, let’s create a table with all England & Belgium Monarchs (Kings and Queens).

Trust me, this serves a point.

Example #2 - Monarchs table

```
INSERT INTO monarchs (id, name, reign_start_year, reign_end_year, country)
(1, 'William I', 1066, 1087, 'Britain'),
(2, 'William II', 1087, 1100, 'Britain'),
(3, 'Henry I', 1100, 1135, 'Britain'),
(4, 'Stephen', 1135, 1154, 'Britain'),
(5, 'Henry II', 1154, 1189, 'Britain'),
(6, 'Richard I', 1189, 1199, 'Britain'),
```

....

```
(38, 'Edward VIII', 1936, 1936, 'Britain'),
(39, 'George VI', 1936, 1952, 'Britain'),
(40, 'Elizabeth II', 1952, 2022, 'Britain'),
(41, 'Charles III', 2022, NULL, 'Britain');
```

```
INSERT INTO monarchs (id, name, reign_start_year, reign_end_year, country)
(42, 'Leopold I', 1831, 1865, 'Belgium'),
(43, 'Leopold II', 1865, 1909, 'Belgium'),
(44, 'Albert I', 1909, 1934, 'Belgium'),
(45, 'Leopold III', 1934, 1951, 'Belgium'),
(46, 'Baudouin', 1951, 1993, 'Belgium'),
(47, 'Albert II', 1993, 2013, 'Belgium'),
(48, 'Philippe', 2013, NULL, 'Belgium');
```

Solve this!

Who is the 3rd monarch in each country?

This should be the result:

name	reign_start_year
Albert I	1909
Henry I	1100

Take a minute to try it out!

Before I reveal the answer - it is based on a real query we have.

As an AgTech company, we store information about fields. Each field, is planted multiple times (at least once a year). Each such instance, is called a cycle.

We need to select the latest cycle for each field. The concept was the same, I used ‘monarchs’ to simplify.


Behold The SQL
Magic

CTE + RANK + Partition

```sql
WITH monarch_ranks AS (
    SELECT
        NAME,
        reign_start_year,
        reign_end_year,
        country,
        RANK() OVER (PARTITION BY country ORDER BY reign_start_year) AS RANK
    FROM
        monarchs
    WHERE
        country IN ('Britain', 'Belgium')
)
SELECT *
FROM monarch_ranks
WHERE RANK = 3;
```

Explanation

- The RANK() function assigns a rank to each monarch based on their ascending order of ascending to the throne.

- The PARTITION BY clause groups the monarchs based on their country, allowing the ranking to be done separately for each country.

Together, they determine the order and ranking of the monarchs within each country, helping us find the third monarch in both.

Elizabeth + Albert = <3

Let's say you want the PREVIOUS monarch in each country (Elizabeth and Albert).

What would you do?

```sql
WITH monarch_ranks AS (
    SELECT
        NAME,
        reign_start_year,
        reign_end_year,
        country,
        RANK() OVER (PARTITION BY country ORDER BY reign_start_year DESC) AS RANK
    FROM
        monarchs
    WHERE
        country IN ('Britain', 'Belgium')
)
SELECT *
FROM monarch_ranks
WHERE RANK = 2;
```

Change the ORDER BY and WHERE.
That's all!

Become an SQL Master

With CTEs

That’s it

Everyone knows basic SQL – Select, DML, DDL. Not many people can use CTEs…

It’s your chance to standout in an interview, or improve your team’s queries!

Follow me for more on software development and team leadership!