Case Statement and Common Table Expression (CTE)

Relevel
by Unacademy

Case Statement and Subqueries

What is a Case Statement?

The case statement goes through several conditions in SQL and returns a value on a met specified condition (if-then-else statement). It is SQL's way of handling if/then logic. We can use a case statement in selected queries along with the 'Where', 'Order By', and 'Group By' clauses.

The CASE statement is followed by at least one pair of WHEN and THEN statements—SQL's equivalent of IF/THEN in Excel. Every CASE statement must end with the END statement. The ELSE statement is optional and provides a way to capture values not specified in the WHEN/THEN statements.

Understanding CASE Statement

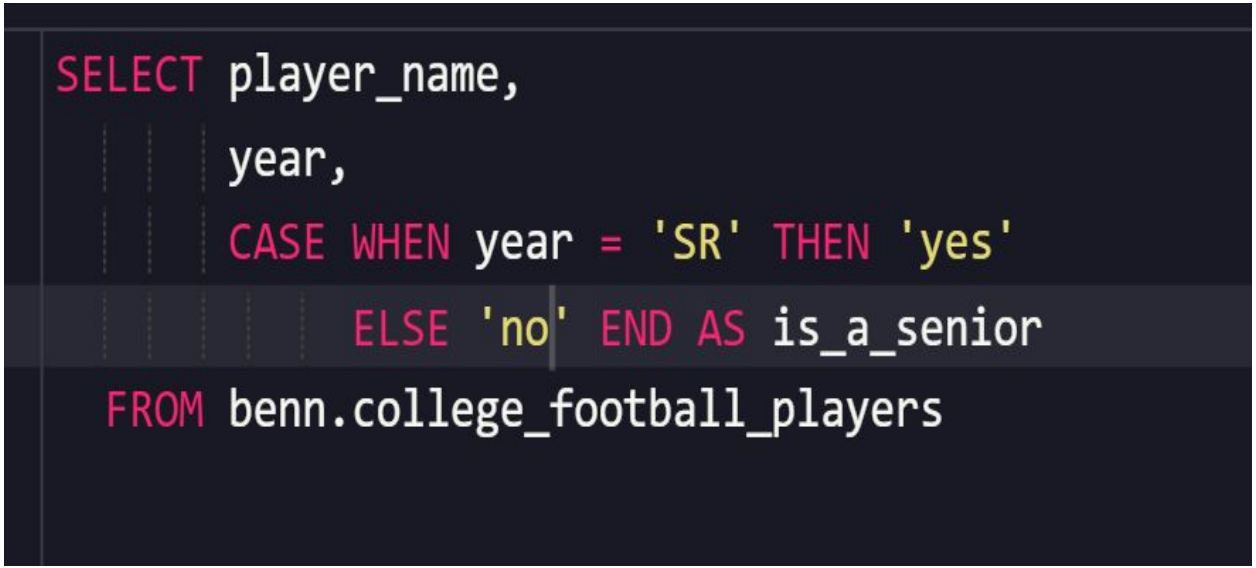
- We will use the below-displayed dataset from mode.com to understand the CASE statement better.

College_football_players

full_school_name	school_name	player_name	position	height	weight	year	hometown	state	id
Cincinnati Bearcats	Cincinnati	Ralph Abernathy	RB	67	161	JR	ATLANTA, GA	GA	1
Cincinnati Bearcats	Cincinnati	Mekale McKay	WR	78	195	SO	LOUISVILLE, KY	KY	2
Cincinnati Bearcats	Cincinnati	Trenier Orr	CB	71	177	SO	WINTER GARDEN, FL	FL	3
Cincinnati Bearcats	Cincinnati	Bennie Coney	QB	75	216	FR	PLANT CITY, FL	FL	4
Cincinnati Bearcats	Cincinnati	Johnny Holton	WR	75	190	JR	MIAMI, FL	FL	5
Cincinnati Bearcats	Cincinnati	Howard Wilder	DB	71	180	JR	SEA ISLAND, GA	GA	6
Cincinnati Bearcats	Cincinnati	Munchie Legaux	QB	77	200	SR	NEW ORLEANS, LA	LA	7
Cincinnati Bearcats	Cincinnati	Mark Barr	WR	73	163	FR	FORT LAUDERDALE, FL	FL	8
Cincinnati Bearcats	Cincinnati	Aaron Brown	CB	71	172	FR	MIAMI, FL	FL	9
Cincinnati Bearcats	Cincinnati	Anthony McClung	WR	73	177	SR	INDIANAPOLIS, IN	IN	10
Cincinnati Bearcats	Cincinnati	Tion Green	RB	73	220	SO	SANFORD, FL	FL	11
Cincinnati Bearcats	Cincinnati	Mike Tyson	S	74	200	SR	NORFOLK, VA	VA	12
Cincinnati Bearcats	Cincinnati	Gunner Kiel	QB	76	208	FR	COLUMBUS, IN	IN	13
Cincinnati Bearcats	Cincinnati	Adrian Witty	S	70	187	JR	DEERFIELD BEACH, FL	FL	14
Cincinnati Bearcats	Cincinnati	Patrick Coyne	FB	73	240	SO	CINCINNATI, OH	OH	15
Cincinnati Bearcats	Cincinnati	Dionne Thrweatt-Vassar	CB	70	190	SR	--	--	16
Cincinnati Bearcats	Cincinnati	Jordan Luallen	FB	75	240	SR	GREENWOOD, IN	IN	17
Cincinnati Bearcats	Cincinnati	Deven Drane	CB	71	187	SR	PLANTATION, FL	FL	18
Cincinnati Bearcats	Cincinnati	Brendon Kay	QB	76	228	SR	MARINE CITY, MI	MI	19
Cincinnati Bearcats	Cincinnati	Leviticus Payne	CB	69	183	SO	SOUTHFIELD, MI	MI	20
Cincinnati Bearcats	Cincinnati	Grant Coleman	CB	71	162	FR	REYNOLDSBURG, OH	OH	21
Cincinnati Bearcats	Cincinnati	Tony Miliano	K	74	186	JR	NORTH BEND, OH	OH	22
Cincinnati Bearcats	Cincinnati	Chris Moore	WR	73	190	SO	TAMPA, FL	FL	23

Understanding CASE Statement

Question - Add a column in database which depicts whether a player is in senior year or not.



```
SELECT player_name,
       year,
       CASE WHEN year = 'SR' THEN 'yes'
            ELSE 'no' END AS is_a_senior
  FROM benn.college_football_players
```

Understanding a simple ‘Case’ Statement Query:

The plain English explanation of the above query is:

- The CASE statement examines each row and checks if the conditional statement—year = 'SR' is accurate.
- If the conditional statement is true, the word "yes" gets recorded in the column named 'is_a_senior' for a given row.
- The word 'no' gets recorded in the column 'is_a_senior' in any row whose conditional statement is false.
- At the same time, when all this is happening, SQL retrieves and displays all the values in the player_name and year columns.

WHY CASE STATEMENTS ARE USED

Case statements are used in analysis majorly: used to look at distribution of a values in a column while performing EDA. Using case statements we can bucket a column to look at distribution. The case statement is a powerful tool you can use when you need to get values based on certain conditions and these case statements are not only confined to EDA but can be extended to data manipulation also.

For example I have a query asking
Add an additional column that displays the player's name if that player is a junior or senior from **Benn.college_football_players** database.

```
SELECT player_name,year,
       CASE WHEN year IN ('JR', 'SR') THEN player_name ELSE NULL END AS upperclass_player_name
  FROM benn.college_football_players
```

Understanding a ‘Case’ Statement with Multiple Conditions:

Question - Group the players according to their weights by adding new column in database.

```
SELECT player_name,
       weight,
       CASE WHEN weight > 250 THEN 'over 250'
            WHEN weight > 200 THEN '201-250'
            WHEN weight > 175 THEN '176-200'
            ELSE '175 or under' END AS weight_group
  FROM benn.college_football_players
```

The above query work as follows:

- Check to see if the weight is greater than 250. If it is, assign 'over 250' in the weight_group column. If it is not greater than 250, move on to the next WHEN/THEN.
- Check to see if the weight is greater than 200. If it is, assign '201-250' in the weight_group column. If not, move on to the next WHEN/THEN.
- Check to see if the weight is greater than 175. Record "176-200" in the weight_group column if it is. If not, record '175 or under'.

CASE Statement with aggregate functions

Question - Group the players on the basis of year to which they belong by adding new column in database.

Query

```
SELECT
        CASE
                WHEN year = 'FR' THEN 'FR'
                WHEN year = 'SO' THEN 'SO'
                WHEN year = 'JR' THEN 'JR'
                WHEN year = 'SR' THEN 'SR'
                ELSE 'No Year Data' END AS year_group,
        COUNT(1) AS count
 FROM college_football_players
GROUP BY CASE WHEN year = 'FR' THEN 'FR'
          WHEN year = 'SO' THEN 'SO'
          WHEN year = 'JR' THEN 'JR'
          WHEN year = 'SR' THEN 'SR'
          ELSE 'No Year Data' END
```

CASE Statement with aggregate functions

The above query work as follows:

- Check to see if year = 'FR'. If true, record 'FR' in year_group. Otherwise, move on to the next WHEN/THEN
- Check to see if year = 'SO'. If true, record 'SO' in year_group. Otherwise, move on to the next WHEN/THEN
- Check to see if year = 'JR'. If true, record 'JR' in year_group. Otherwise, move on to the next WHEN/THEN
- Check to see if year = 'SR'. If true, record 'SR' in year_group. Otherwise, put 'No Year Data' in the year_group
- Then count the number of players in each year_group category.

CASE Statement with aggregate functions

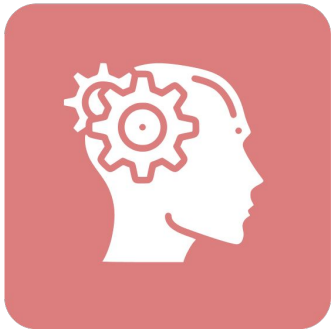
The above query can be written in a better format.

Query

```
SELECT
        CASE
                WHEN year = 'FR' THEN 'FR'
                WHEN year = 'SO' THEN 'SO'              WHEN
year = 'JR' THEN 'JR'
                WHEN year = 'SR' THEN 'SR'
                ELSE 'No Year Data' END AS year_group,
        COUNT(1) AS count
 FROM college_football_players
GROUP BY 1
```

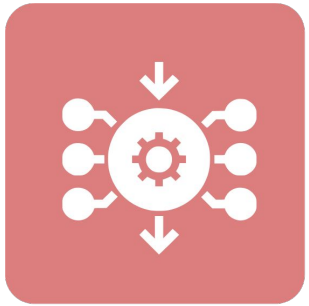
Recap of Case Statement:

- The CASE statement is always recorded in the 'SELECT' clause.
- CASE must constitute WHEN, THEN, and END components. 'ELSE' component is optional.
- You can generate any conditional statement using any conditional operator (like WHERE) in the middle of WHEN and THEN. This includes linking together multiple conditional statements using AND and OR.
- You can include multiple WHEN statements and an ELSE statement to deal with any unaddressed conditions.



Common Table Expression (CTE)

Common table expressions (CTEs) are an SQL functionality that allows you to perform complex, multi-step transformations in a single, easy-to-read query. They are a helpful tool for beginners and experts alike because of their power, readability, and flexibility.



How do they work?

If we put this simply, CTE allows you to create temporary datasets that you can refer to in the future for a query.

These temporary datasets are "available" to use for the duration of the query itself, but they are not stored in your database. They are extinct once your query has been implemented.

Why do we need them?

At their core, CTEs perform two things:

1. CTEs solve what I like to call the "logic on top of logic" problem. This takes place when you have to perform data manipulation and use the resulting dataset to perform extensive manipulation.
2. CTEs makes your code much more readable and simpler to work with.

What does the syntax look like?

The syntax for CTEs is uncomplicated:

1. Begin with typing the keyword "with" + any name you'd like to use to refer to the dataset you create + "as".
2. Write the query you'd like to perform in parenthesis.
3. Write other queries that refer to the CTE.

Syntax of CTE

with my_cte as -- call for creating cte

(select * from the table) -- putting data into CTE

select * from my_cte; -- Querying CTE

Understanding CTE through examples:

We will understand the concept through this table.

employee_id	first_name	last_name	position	outlet	region	bonus
1	Max	Black	manager	123	South	2305.45
2	Jane	Wolf	cashier	123	South	1215.35
3	Kate	White	customer service specialist	123	South	1545.75
4	Andrew	Smart	customer service specialist	123	South	1800.55
5	John	Ruder	manager	105	South	2549.45
6	Sebastian	Cornell	cashier	105	South	1505.25
7	Diana	Johnson	customer service specialist	105	South	2007.95
8	Sofia	Blanc	manager	224	North	2469.75
9	Jack	Spider	customer service specialist	224	North	2100.50
10	Maria	Le	cashier	224	North	1325.65
11	Anna	Winfrey	manager	211	North	2390.25
12	Marion	Spencer	cashier	211	North	1425.25

CTE with Join

Write a query to add a column average_bonus_for_position in the above dataset. average_bonus_for_position in the above dataset is the average bonus for a given position(e.g.cashier, manager) irrespective of the individual.

Answer:

```
WITH avg_position AS (
    SELECT position, AVG(bonus) AS average_bonus_for_position
    FROM snt2017.employee_bonus
    GROUP BY position)
SELECT b.employee_id, b.first_name, b.last_name, b.position, b.bonus,
ap.average_bonus_for_position
FROM snt2017.employee_bonus b
JOIN avg_position ap
ON b.position = ap.position;
```

CTE with Join

✓ 12 rows | 576B returned in 588ms

	employee_id	first_name	last_name	position	bonus	average_bonus_for_position
1	1	Max	Black	manager	2305.45	2428.725
2	2	Jane	Wolf	cashier	1215.35	1367.875
3	3	Kate	White	customer service specialist	1545.75	1863.6875
4	4	Andrew	Smart	customer service specialist	1800.55	1863.6875
5	5	John	Ruder	manager	2549.45	2428.725
6	6	Sebastian	Cornell	cashier	1505.25	1367.875
7	7	Diana	Johnson	customer service specialist	2007.95	1863.6875
8	8	Sofia	Blanc	manager	2469.75	2428.725
9	9	Jack	Spider	customer service specialist	2100.5	1863.6875
10	10	Maria	Le	cashier	1325.65	1367.875
11	11	Anna	Winfrey	manager	2390.25	2428.725
12	12	Marion	Spencer	cashier	1425.25	1367.875

Multiple CTE:

Question: Write a query to add two columns average_bonus_for_position and average_bonus_for_region in the above dataset. average_bonus_for_position in the above dataset is the average bonus for a given position(e.g. cashier, manager) irrespective of the individual. average_bonus_for_region in the above dataset is the average bonus for a given position(e.g. north, south) irrespective of the individual.

Answer:

```
WITH avg_position AS (
   SELECT position, AVG(bonus) AS average_bonus_for_position
   FROM snt2017.employee_bonus
GROUP BY position),
   avg_region AS (
   SELECT region, AVG (bonus) AS average_bonus_for_region
   FROM snt2017.employee_bonus
   GROUP BY region)
SELECT b.employee_id, b.first_name, b.last_name, b.position, b.region, b.bonus, ap.average_bonus_for_position, ar.average_bonus_for_region
FROM snt2017.employee_bonus b
JOIN avg_position ap
ON b.position = ap.position
JOIN avg_region ar
ON b.region = ar.region;
```

Multiple CTE:

Query 1

12 rows | 732B returned in 469ms

Copy

	employee_id	first_name	last_name	position	region	bonus	average_bonus_for_position	average_bonus_for_region
1	1	Max	Black	manager	South	2305.45	2428.725	1847.1071
2	2	Jane	Wolf	cashier	South	1215.35	1367.875	1847.1071
3	3	Kate	White	customer service specialist	South	1545.75	1863.6875	1847.1071
4	4	Andrew	Smart	customer service specialist	South	1800.55	1863.6875	1847.1071
5	5	John	Ruder	manager	South	2549.45	2428.725	1847.1071
6	6	Sebastian	Cornell	cashier	South	1505.25	1367.875	1847.1071
7	7	Diana	Johnson	customer service specialist	South	2007.95	1863.6875	1847.1071
8	8	Sofia	Blanc	manager	North	2469.75	2428.725	1942.28
9	9	Jack	Spider	customer service specialist	North	2100.5	1863.6875	1942.28
10	10	Maria	Le	cashier	North	1325.65	1367.875	1942.28
11	11	Anna	Winfrey	manager	North	2390.25	2428.725	1942.28
12	12	Marion	Spencer	cashier	North	1425.25	1367.875	1942.28

Practice Questions:

Instructions:

- We will use mode.com for all the practice questions
- We will use the following datasets in the questions below:
    - benn.college_football_players

Question-1:

Question-1: Write a query that includes a column flagged "YES" when a player is from California, and classify the results with those players first.

Question-1:

Answer-1:

```
SELECT player_name,
      state,
      CASE WHEN state = 'CA' THEN 'yes'
          ELSE NULL END AS from_california
 FROM benn.college_football_players
ORDER BY 3
```

Question-1:

```sql
SELECT player_name,
       state,
       CASE WHEN state = 'CA' THEN 'yes'
            ELSE NULL END AS from_california
  FROM benn.college_football_players
 ORDER BY 3
```

	player_name	state	from_california
1	David Irving	CA	yes
2	Sadale Foster	CA	yes
3	Isaac Goins	CA	yes
4	Brett Medders	CA	yes
5	Julian Winters	CA	yes
6	Will Smith	CA	yes
7	Cayman Carter	CA	yes
8	Deion Williams	CA	yes
9	Eric Morris	CA	yes
10	Ben Loth	CA	yes
11	Aaron Bennett	CA	yes
12	Brennan Clay	CA	yes
13	Mason Orradre	CA	yes
14	Geoff Swaim	CA	yes
15	Chris Gudmunson	CA	yes
16	Daniel Roundtree	CA	yes
17	Spencer Hollie	CA	yes
18	Phillip Carter	CA	yes
19	Asante Cleveland	CA	yes
20	Eric Jackson	CA	yes

Question-2:

Question-2: Write a query that includes players' names and a column that classifies them into four categories based on height. Remember that the answer we provide is only one of many possible answers since you could divide players' heights in many ways.

Question-2:

Answer-2:

```
SELECT player_name,
       height,
       CASE WHEN height > 74 THEN 'over 74'
            WHEN height > 72 THEN '73-74'
            WHEN height > 70 THEN '71-72'
            ELSE 'under 70' END AS height_group
  FROM benn.college_football_players
```

Question-2:

```sql
SELECT player_name,
       height,
       CASE WHEN height > 74 THEN 'over 74'
            WHEN height > 72 THEN '73-74'
            WHEN height > 70 THEN '71-72'
            ELSE 'under 70' END AS height_group
  FROM benn.college_football_players
```

	player_name	height	height_group
1	Ralph Abernathy	67	under 70
2	Mekale McKay	78	over 74
3	Trenier Orr	71	71-72
4	Bennie Coney	75	over 74
5	Johnny Holton	75	over 74
6	Howard Wilder	71	71-72
7	Munchie Legaux	77	over 74
8	Mark Barr	73	73-74
9	Aaron Brown	71	71-72
10	Anthony McClung	73	73-74

Question-3:

Question-3: Write a query that selects all columns from benn.college_football_players and adds an additional column that displays the player's name if junior or senior.

Answer-3:

```
SELECT *,
      CASE WHEN year IN ('JR', 'SR') THEN player_name ELSE NULL END AS upperclass_player_name
  FROM benn.college_football_players
```

Question-3:

```
SELECT *,
       CASE WHEN year IN ('JR', 'SR') THEN player_name ELSE NULL END AS upperclass_player_name
  FROM benn.college_football_players
```

	full_school_name	school_name	player_name	position	height	weight	year	hometown	state
1	Cincinnati Bearcats	Cincinnati	Ralph Abernathy	RB	67	161	JR	ATLANTA, GA	GA
2	Cincinnati Bearcats	Cincinnati	Mekale McKay	WR	78	195	SO	LOUISVILLE, KY	KY
3	Cincinnati Bearcats	Cincinnati	Trenier Orr	CB	71	177	SO	WINTER GARDEN, FL	FL
4	Cincinnati Bearcats	Cincinnati	Bennie Coney	QB	75	216	FR	PLANT CITY, FL	FL
5	Cincinnati Bearcats	Cincinnati	Johnny Holton	WR	75	190	JR	MIAMI, FL	FL
6	Cincinnati Bearcats	Cincinnati	Howard Wilder	DB	71	180	JR	SEA ISLAND, GA	GA
7	Cincinnati Bearcats	Cincinnati	Munchie Legaux	QB	77	200	SR	NEW ORLEANS, LA	LA
8	Cincinnati Bearcats	Cincinnati	Mark Barr	WR	73	163	FR	FORT LAUDERDALE, FL	FL
9	Cincinnati Bearcats	Cincinnati	Aaron Brown	CB	71	172	FR	MIAMI, FL	FL
10	Cincinnati Bearcats	Cincinnati	Anthony McClung	WR	73	177	SR	INDIANAPOLIS, IN	IN
11	Cincinnati Bearcats	Cincinnati	Tion Green	RB	73	220	SO	SANFORD, FL	FL
12	Cincinnati Bearcats	Cincinnati	Mike Tyson	S	74	200	SR	NORFOLK, VA	VA
13	Cincinnati Bearcats	Cincinnati	Gunner Kiel	QB	76	208	FR	COLUMBUS, IN	IN
14	Cincinnati Bearcats	Cincinnati	Adrian Witty	S	70	187	JR	DEERFIELD BEACH, FL	FL
15	Cincinnati Bearcats	Cincinnati	Patrick Coyne	FB	73	240	SO	CINCINNATI, OH	OH
16	Cincinnati Bearcats	Cincinnati	Dionne Thrweatt-Vass...	CB	70	190	SR	--	--
17	Cincinnati Bearcats	Cincinnati	Jordan Luallen	FB	75	240	SR	GREENWOOD, IN	IN
18	Cincinnati Bearcats	Cincinnati	Deven Drane	CB	71	187	SR	PLANTATION, FL	FL
19	Cincinnati Bearcats	Cincinnati	Brendon Kay	QB	76	228	SR	MARINE CITY, MI	MI
20	Cincinnati Bearcats	Cincinnati	Leviticus Payne	CB	69	183	SO	SOUTHFIELD, MI	MI
21	Cincinnati Bearcats	Cincinnati	Grant Coleman	CB	71	162	FR	REYNOLDSBURG, OH	OH
22	Cincinnati Bearcats	Cincinnati	Tony Miliano	K	74	186	JR	NORTH BEND, OH	OH
23	Cincinnati Bearcats	Cincinnati	Chris Moore	WR	73	190	SO	TAMPA, FL	FL
24	Cincinnati Bearcats	Cincinnati	Michael Colosimo	QB	75	216	JR	FORT MITCHELL, KY	KY

Question-4:

Question-4: Write a query that counts the number of 300lb+ players for each of these regions: West Coast (CA, OR, WA), Texas, and Others.

Answer-4:

```
SELECT CASE WHEN state IN ('CA', 'OR', 'WA') THEN 'West Coast'
            WHEN state = 'TX' THEN 'Texas'
            ELSE 'Other' END AS arbitrary_regional_designation,
            COUNT(1) AS players
  FROM benn.college_football_players
 WHERE weight >= 300
 GROUP BY 1
```

Question-4:

```
SELECT CASE WHEN state IN ('CA', 'OR', 'WA') THEN 'West Coast'
            WHEN state = 'TX' THEN 'Texas'
            ELSE 'Other' END AS arbitrary_regional_designation,
            COUNT(1) AS players
  FROM benn.college_football_players
 WHERE weight >= 300
 GROUP BY 1
```

3 rows | 44B returned in 548ms

arbitrary_regional_designation	players
Other	1590
West Coast	186
Texas	208

Question-5:

Question-5: Write a query that calculates the combined weight of all underclass players (FR/SO) in California and the combined weight of all upperclass players (JR/SR) in California.

Answer-5:

```
SELECT CASE WHEN year IN ('FR', 'SO') THEN 'underclass'
            WHEN year IN ('JR', 'SR') THEN 'upperclass'
            ELSE NULL END AS class_group,
       SUM(weight) AS combined_player_weight
  FROM benn.college_football_players
 WHERE state = 'CA'
 GROUP BY 1
```

Question-5:

```
SELECT CASE WHEN year IN ('FR', 'SO') THEN 'underclass'
            WHEN year IN ('JR', 'SR') THEN 'upperclass'
            ELSE NULL END AS class_group,
       SUM(weight) AS combined_player_weight
  FROM benn.college_football_players
 WHERE state = 'CA'
 GROUP BY 1
```

2 rows | 36B returned in 421ms

class_group	combined_player_weight
underclass	274374
upperclass	262452

Question-6:

Question-6: Write a query that shows the number of players in each state, with FR, SO, JR, and SR players in different columns, as well as another column for the total number of players. Order results such as states with the most players come first.

Answer-6:

```
SELECT state,
      COUNT(CASE WHEN year = 'FR' THEN 1 ELSE NULL END) AS fr_count,
      COUNT(CASE WHEN year = 'SO' THEN 1 ELSE NULL END) AS so_count,
      COUNT(CASE WHEN year = 'JR' THEN 1 ELSE NULL END) AS jr_count,
      COUNT(CASE WHEN year = 'SR' THEN 1 ELSE NULL END) AS sr_count,
      COUNT(1) AS total_players
 FROM benn.college_football_players
GROUP BY state
ORDER BY total_players DESC
```

Question-6:

```sql
SELECT state,
       COUNT(CASE WHEN year = 'FR' THEN 1 ELSE NULL END) AS fr_count,
       COUNT(CASE WHEN year = 'SO' THEN 1 ELSE NULL END) AS so_count,
       COUNT(CASE WHEN year = 'JR' THEN 1 ELSE NULL END) AS jr_count,
       COUNT(CASE WHEN year = 'SR' THEN 1 ELSE NULL END) AS sr_count,
       COUNT(1) AS total_players
  FROM benn.college_football_players
 GROUP BY state
 ORDER BY total_players DESC
```

✓ 67 rows | 3KB returned in 848ms

	state	fr_count	so_count	jr_count	sr_count	total_players
1	TX	1055	639	649	498	2841
2	FL	944	571	523	497	2535
3	CA	769	478	632	541	2420
4	GA	657	418	368	315	1758
5	OH	443	284	229	207	1163
6	NC	442	248	187	137	1014
7	PA	311	205	189	210	915
8	IL	350	204	185	144	883
9	LA	319	210	178	164	871
10	VA	344	169	176	162	851
11	NJ	232	195	210	187	824
12	AL	299	182	175	112	768
13	--	335	99	102	177	713
14	TN	274	148	150	112	684
15	NY	183	145	138	165	631
16	MD	186	141	144	132	603
17	SC	190	134	100	160	584
18	MI	189	90	87	79	445

Question-7:

Question-7: Write a query that shows the number of players at schools with the names beginning with A through M and the number of schools with the names beginning with N - Z.

Answer-7:

```
SELECT CASE WHEN school_name < 'n' THEN 'A-M'
            WHEN school_name >= 'n' THEN 'N-Z'
            ELSE NULL END AS school_name_group,
       COUNT(1) AS players
  FROM benn.college_football_players
 GROUP BY 1
```

Question-7:

```
SELECT CASE WHEN school_name < 'n' THEN 'A-M'
            WHEN school_name >= 'n' THEN 'N-Z'
            ELSE NULL END AS school_name_group,
       COUNT(1) AS players
  FROM benn.college_football_players
 GROUP BY 1
```

2 rows | 22B returned in 498ms

school_name_group	players
N-Z	12512
A-M	13786

In the next class we will study:

- Window functions - I

THANK YOU