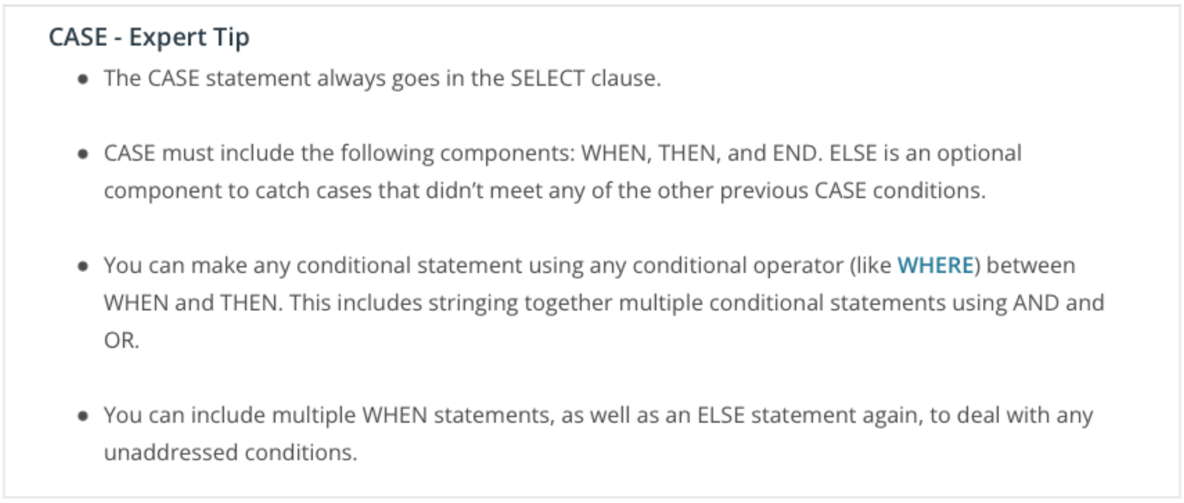
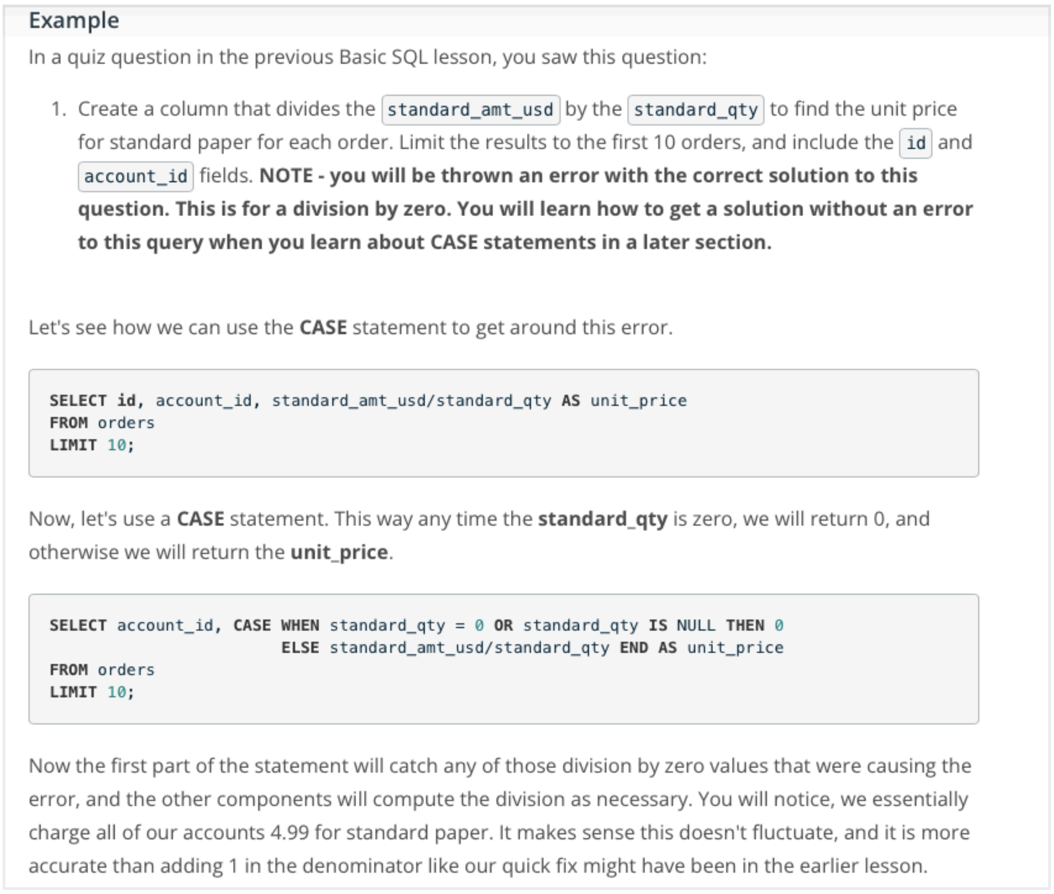
# SQL CASES



### CASE - Expert Tip

- The CASE statement always goes in the SELECT clause.
- CASE must include the following components: WHEN, THEN, and END. ELSE is an optional component to catch cases that didn't meet any of the other previous CASE conditions.
- You can make any conditional statement using any conditional operator (like **WHERE**) between WHEN and THEN. This includes stringing together multiple conditional statements using AND and OR.
- You can include multiple WHEN statements, as well as an ELSE statement again, to deal with any unaddressed conditions.



### Example

In a quiz question in the previous Basic SQL lesson, you saw this question:

1. Create a column that divides the `standard_amt_usd` by the `standard_qty` to find the unit price for standard paper for each order. Limit the results to the first 10 orders, and include the `id` and `account_id` fields. **NOTE - you will be thrown an error with the correct solution to this question. This is for a division by zero. You will learn how to get a solution without an error to this query when you learn about CASE statements in a later section.**

Let's see how we can use the **CASE** statement to get around this error.

```
SELECT id, account_id, standard_amt_usd/standard_qty AS unit_price
FROM orders
LIMIT 10;
```

Now, let's use a **CASE** statement. This way any time the **standard_qty** is zero, we will return 0, and otherwise we will return the **unit_price**.

```
SELECT account_id, CASE WHEN standard_qty = 0 OR standard_qty IS NULL THEN 0
                        ELSE standard_amt_usd/standard_qty END AS unit_price
FROM orders
LIMIT 10;
```

Now the first part of the statement will catch any of those division by zero values that were causing the error, and the other components will compute the division as necessary. You will notice, we essentially charge all of our accounts 4.99 for standard paper. It makes sense this doesn't fluctuate, and it is more accurate than adding 1 in the denominator like our quick fix might have been in the earlier lesson.

## Solutions: CASE

1. Write a query to display for each order, the account ID, total amount of the order, and the level of the order - 'Large' or 'Small' - depending on if the order is $3000 or more, or less than $3000.

```
SELECT account_id, total_amt_usd,
CASE WHEN total_amt_usd > 3000 THEN 'Large'
ELSE 'Small' END AS order_level
FROM orders;
```

2. Write a query to display the number of orders in each of three categories, based on the `total` number of items in each order. The three categories are: 'At Least 2000', 'Between 1000 and 2000' and 'Less than 1000'.

```
SELECT CASE WHEN total >= 2000 THEN 'At Least 2000'
   WHEN total >= 1000 AND total < 2000 THEN 'Between 1000 and 2000'
   ELSE 'Less than 1000' END AS order_category,
COUNT(*) AS order_count
FROM orders
GROUP BY 1;
```

3. We would like to understand 3 different branches of customers based on the amount associated with their purchases. The top branch includes anyone with a Lifetime Value (total sales of all orders) `greater than 200,000` usd. The second branch is between `200,000 and 100,000` usd. The lowest branch is anyone `under 100,000` usd. Provide a table that includes the **level** associated with each **account**. You should provide the **account name**, the **total sales of all orders** for the customer, and the **level**. Order with the top spending customers listed first.

```
SELECT a.name, SUM(total_amt_usd) total_spent,
     CASE WHEN SUM(total_amt_usd) > 200000 THEN 'top'
     WHEN  SUM(total_amt_usd) > 100000 THEN 'middle'
     ELSE 'low' END AS customer_level
FROM orders o
JOIN accounts a
ON o.account_id = a.id
GROUP BY a.name
ORDER BY 2 DESC;
```

4. We would now like to perform a similar calculation to the first, but we want to obtain the total amount spent by customers only in `2016` and `2017`. Keep the same **level**s as in the previous question. Order with the top spending customers listed first.

```
SELECT a.name, SUM(total_amt_usd) total_spent,
     CASE WHEN SUM(total_amt_usd) > 200000 THEN 'top'
     WHEN  SUM(total_amt_usd) > 100000 THEN 'middle'
     ELSE 'low' END AS customer_level
FROM orders o
JOIN accounts a
ON o.account_id = a.id
WHERE occurred_at > '2015-12-31'
GROUP BY 1
ORDER BY 2 DESC;
```