SQL How To:
Stack different levels of aggregation (4 methods)

Suppose we have some sales data

```
1   SELECT category, sub_category, product, SalesAmount
2   FROM vSales;
```

	category	sub_category	product	SalesAmount
1	Bikes	Road Bikes	Road-150 Red, 62	3578.27
2	Bikes	Mountain Bikes	Mountain-100 Silver, 44	3399.99
3	Bikes	Mountain Bikes	Mountain-100 Silver, 44	3399.99
4	Bikes	Road Bikes	Road-650 Black, 62	699.0982
5	Bikes	Mountain Bikes	Mountain-100 Silver, 44	3399.99
6	Bikes	Road Bikes	Road-150 Red, 44	3578.27
7	Bikes	Road Bikes	Road-150 Red, 62	3578.27
8	Bikes	Mountain Bikes	Mountain-100 Black, 48	3374.99
9	Bikes	Mountain Bikes	Mountain-100 Silver, 38	3399.99
10	Bikes	Road Bikes	Road-150 Red, 48	3578.27

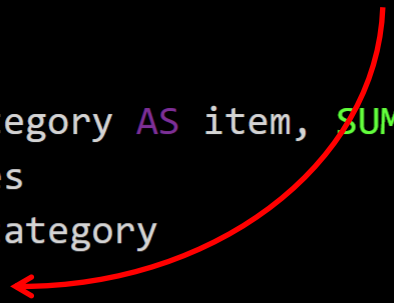
And we'd like to sum sales by each level
And return a two-column table like this:

	item	sales
1	Clothing	339772.61
2	Bikes	28318144.6507
3	Accessories	700759.96
4	Cleaners	7218.60
5	Hydration Packs	40307.67
6	Road Bikes	14520584.0363
7	Bottles and Cages	56798.19
8	Vests	35687.00
9	Helmets	225335.60
10	Bike Stands	39591.00

This result contains SUM of sales by category, sub_category and product, stacked on top of each other

Method 1: UNION ALL

We write three queries and connect them with the UNION ALL set operator



```
1   SELECT category AS item, SUM(SalesAmount) AS sales
2   FROM vSales
3   GROUP BY category
4   UNION ALL
5   SELECT sub_category AS item, SUM(SalesAmount) AS sales
6   FROM vSales
7   GROUP BY sub_category
8   UNION ALL
9   SELECT product AS item, SUM(SalesAmount) AS sales
10  FROM vSales
11  GROUP BY product;
```

Method 2: ROLLUP and COALESCE (a) – inspecting ROLLUP

ROLLUP goes in the GROUP BY clause and we wrap the columns we want to roll up in parentheses

```
1   SELECT category, sub_category, product, SUM(SalesAmount) AS sales
2   FROM vSales
3   GROUP BY ROLLUP (category,sub_category,product);
```

	category	sub_category	product	sales	
28	Accessories	Tires and Tubes	Touring Tire	27105.65	product
29	Accessories	Tires and Tubes	Touring Tire Tu...	7425.12	product
30	Accessories	Tires and Tubes	*NULL*	245529.32	sub_category
31	Accessories	*NULL*	*NULL*	700759.96	category

Method 2: ROLLUP and COALESCE (b) – adding COALESCE

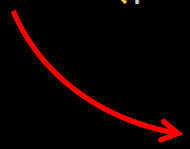
```
1   SELECT category,
2       sub_category,
3       product,
4       COALESCE(product,sub_category,category) AS item,
5       SUM(SalesAmount) AS sales
6   FROM vSales
7   GROUP BY ROLLUP (category,sub_category,product)
```

We combine the three columns into one using COALESCE on the columns in reverse order

	category	sub_category	product	item	sales
28	Accessories	Tires and Tubes	Touring Tire	Touring Tire	27105.65
29	Accessories	Tires and Tubes	Touring Tire Tu...	Touring Tire Tube	7425.12
30	Accessories	Tires and Tubes	*NULL*	Tires and Tubes	245529.32
31	Accessories	*NULL*	*NULL*	Accessories	700759.96

Method 2: ROLLUP and COALESCE (c) – final query

```sql
SELECT
    COALESCE(product,sub_category,category) AS item,
    SUM(SalesAmount) AS sales
FROM vSales
GROUP BY ROLLUP (category,sub_category,product)
HAVING COALESCE(product,sub_category,category) IS NOT NULL;
```



ROLLUP in this form has created a grand total with NULL in each column, meaning the COALESCE is also NULL. We remove it in the HAVING clause.

Method 3: GROUPING SETS
(a) – inspecting GROUPING SETS

We define GROUPING SETS in GROUP BY. Each set is enclosed in parentheses

```
1    SELECT category, sub_category, product, SUM(SalesAmount) AS sales
2    FROM vSales
3    GROUP BY GROUPING SETS ((category),(sub_category),(product));
```

	category	sub_category	product	sales	
128	*NULL*	*NULL*	Women's Mountain Sh...	25406.37	product
129	*NULL*	*NULL*	Women's Mountain Sh...	24636.48	product
130	*NULL*	*NULL*	Women's Mountain Sh...	21276.96	product
131	*NULL*	Bike Racks	*NULL*	39360.00	sub_category
132	*NULL*	Bike Stands	*NULL*	39591.00	sub_category
133	*NULL*	Bottles and Ca...	*NULL*	56798.19	sub_category

Method 3: GROUPING SETS (b) – final query

We add COALESCE in the same way as ROLLUP

```
1  SELECT
2      COALESCE(product,sub_category,category) AS item,
3      SUM(SalesAmount) AS sales
4  FROM vSales
5  GROUP BY GROUPING SETS ((category),(sub_category),(product))
```

We don't need the HAVING clause because we haven't told GROUPING SETS to create a grand total

```
5  GROUP BY GROUPING SETS ((),(category),(sub_category),(product))
```

Method 4: CROSS JOIN and CHOOSE

CHOOSE is a SQL Server function. MySQL calls it ELT

CHOOSE uses the first parameter (an integer) to decide which of the other parameters to return

```
1   SELECT
2       CHOOSE(rh.i,category,sub_category,product) AS item,
3       SUM(SalesAmount) AS sum_sales_amount
4   FROM vSales, (VALUES (1),(2),(3)) AS rh(i)
5   GROUP BY
6       CHOOSE(rh.i,category,sub_category,product);
7
```

A comma can be used as a shortcut for CROSS JOIN

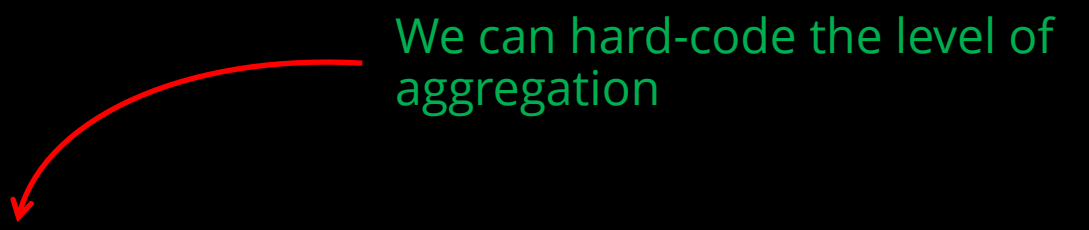
The FROM clause duplicates each row in vSales three times. Each duplicate is associated with one of the three numbers in the VALUES table aliased “rh”.

BUT WAIT! Which rows are categories, which are sub-categories and which are products?!

	item	sales
1	Clothing	339772.61
2	Bikes	28318144.6507
3	Accessories	700759.96
4	Cleaners	7218.60
5	Hydration Packs	40307.67
6	Road Bikes	14520584.0363
7	Bottles and Cages	56798.19
8	Vests	35687.00
9	Helmets	225335.60
10	Bike Stands	39591.00

We're not done...

Method 1: UNION ALL




```sql
SELECT 'category' AS item_type, category AS item, SUM(SalesAmount) AS sales
FROM vSales
GROUP BY category
UNION ALL
SELECT 'sub_category' AS item_type, sub_category AS item, SUM(SalesAmount) AS sales
FROM vSales
GROUP BY sub_category
UNION ALL
SELECT 'product' AS item_type, product AS item, SUM(SalesAmount) AS sales
FROM vSales
GROUP BY product;
```

Method 2: ROLLUP and COALESCE (a) – the GROUPING function

```sql
SELECT
    GROUPING(category),
    GROUPING(sub_category),
    GROUPING(product),
    category,
    sub_category,
    product,
    SUM(SalesAmount) AS sales
FROM vSales
GROUP BY ROLLUP (category,sub_category,product);
```

The GROUPING function returns 1 if ROLLUP has created a NULL, 0 otherwise

	(N...	(...	(...	category	sub_category	product	sales
29	0	0	0	Accessories	Tires and Tubes	Touring Tire Tu...	7425.12
30	0	0	1	Accessories	Tires and Tubes	*NULL*	245529.32
31	0	1	1	Accessories	*NULL*	*NULL*	700759.96

Method 2: ROLLUP and COALESCE (b) – final query

We can use CHOOSE and the sum of the GROUPING functions to determine which value to return in the item_type column

```
1   SELECT  CHOOSE(
2               GROUPING(category)+GROUPING(sub_category)+GROUPING(product)+1,
3               'product',
4               'sub_category',
5               'category'
6           ) AS item_type,
7           COALESCE(product,sub_category,category) AS item,
8           SUM(SalesAmount) AS sales
9   FROM vSales
10  GROUP BY ROLLUP (category,sub_category,product)
11  HAVING COALESCE(product,sub_category,category) IS NOT NULL;
```

Method 3: GROUPING SETS

The GROUPING function works in the same way for GROUPING SETS as it does for ROLLUP

```
1   SELECT  CHOOSE(
2               GROUPING(category)+GROUPING(sub_category)+GROUPING(product)+1,
3               'product',
4               'sub_category',
5               'category'
6           ) AS item_type,
7           COALESCE(product,sub_category,category) AS item,
8           SUM(SalesAmount) AS sales
9   FROM vSales
10  GROUP BY GROUPING SETS ((category),(sub_category),(product))
```

Method 4: CROSS JOIN and CHOOSE

CHOOSE is a SQL Server function. MySQL calls it ELT

We just use CHOOSE again, but return the column names as text

```
1   SELECT
2       CHOOSE(rh.i,'category','sub_category','product') AS item_type,
3       CHOOSE(rh.i,category,sub_category,product) AS item,
4       SUM(SalesAmount) AS sum_sales_amount
5   FROM vSales, (VALUES (1),(2),(3)) AS rh(i)
6   GROUP BY
7       rh.i, CHOOSE(rh.i,category,sub_category,product);
```

We can just add this to GROUP BY instead of the entire expression for item_type because the remaining parameters of the expression are literals

That’s 4 methods.

There are probably others.

Always consider:

- Performance
- Maintenance
- Your team
- Your sanity