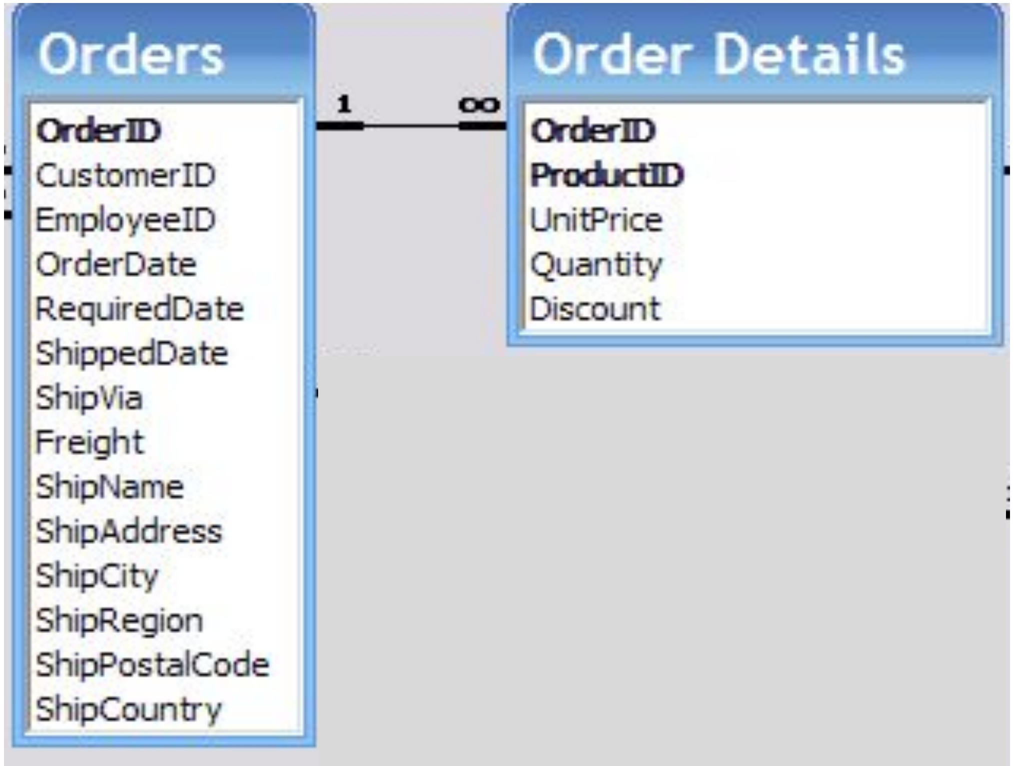
ORIE 3120

Lecture 4: SQL #3 [GROUP BY]

GROUP BY

Suppose I want to know how much was paid for each order

- Orders doesn’t have information on how much customers paid
- OrderDetail does (UnitPrice, Quantity, Discount), but there is a record for each product in an order, not for the whole order

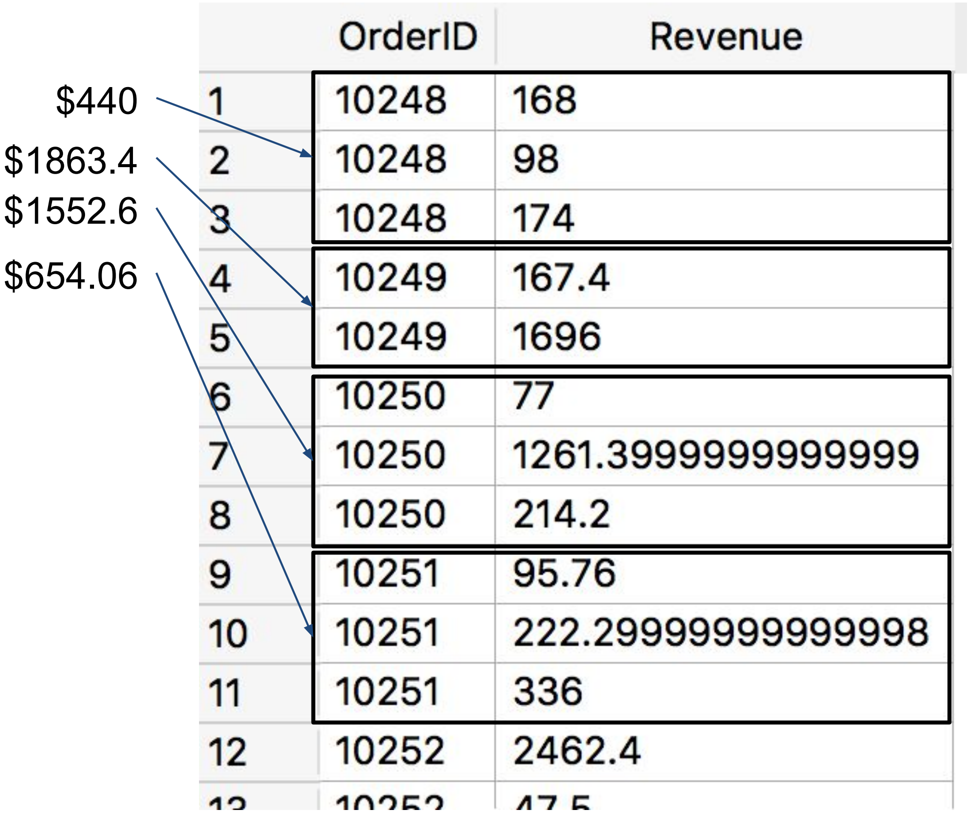


Here’s a good start

```
SELECT  OrderID,
        UnitPrice*Quantity*(1-Discount) AS Revenue
FROM OrderDetails
ORDER BY OrderID
```

Records with the same orderID are next to each other because of the ORDER BY.

For each block of records with the same orderID in this query result, I want to sum up the revenue.



	OrderID	Revenue
1	10248	168
2	10248	98
3	10248	174
4	10249	167.4
5	10249	1696
6	10250	77
7	10250	1261.3999999999999
8	10250	214.2
9	10251	95.76
10	10251	222.29999999999998
11	10251	336
12	10252	2462.4
13	10253	47.5

$440

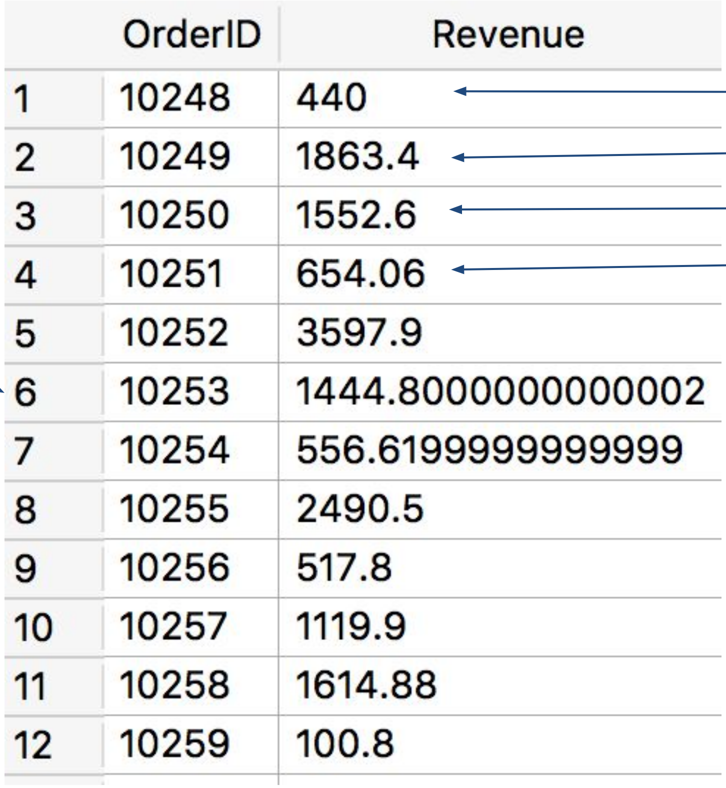
$1863.4

$1552.6

$654.06

SQL can aggregate records by OrderID and sum them

```
SELECT   OrderID,
         SUM(UnitPrice*Quantity*(1-Discount)) AS Revenue
FROM OrderDetails
GROUP BY OrderID
```



	OrderID	Revenue
1	10248	440
2	10249	1863.4
3	10250	1552.6
4	10251	654.06
5	10252	3597.9
6	10253	1444.8000000000002
7	10254	556.6199999999999
8	10255	2490.5
9	10256	517.8
10	10257	1119.9
11	10258	1614.88
12	10259	100.8

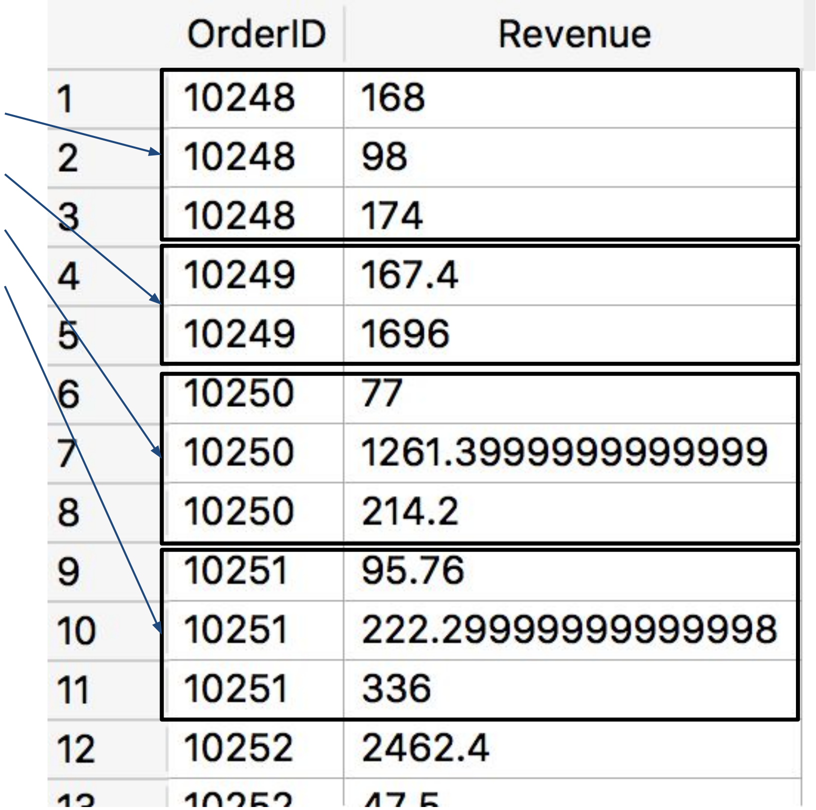
$440

$1863.4

$1552.6

$654.06

Result from query on the previous slide



	OrderID	Revenue
1	10248	168
2	10248	98
3	10248	174
4	10249	167.4
5	10249	1696
6	10250	77
7	10250	1261.3999999999999
8	10250	214.2
9	10251	95.76
10	10251	222.29999999999998
11	10251	336
12	10252	2462.4
13	10252	47.5

How to use GROUP BY

Syntax:

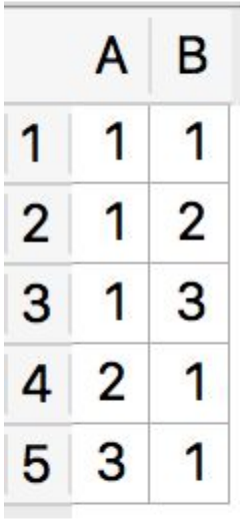
```
SELECT A, SUM(B) FROM T GROUP BY A
```

For each value of A in the table, GROUP BY:

- Finds all records with that value of A
- Compute the sum of field B for those records

Example

Table T



	A	B
1	1	1
2	1	2
3	1	3
4	2	1
5	3	1

What records does this query produce?
SELECT A, SUM(B) FROM T GROUP BY A

(a)

A	SUM(B)
1	1
1	2
1	3
2	1
3	1

(b)

A	SUM(B)
1	6
2	1
3	1

(d)

A	SUM(B)
8	8

(c)

A	SUM(B)
6	1
1	2
1	3

(e)

A	SUM(B)
1	8

GROUP BY can do things beyond SUM

SQLite supports these aggregation functions:

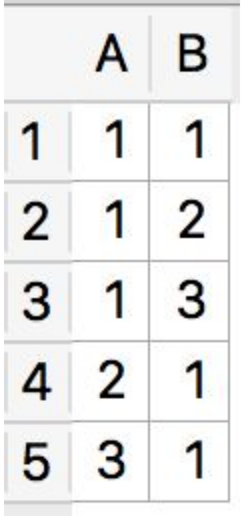
- SUM: sum of the aggregated records
- COUNT: number of aggregated records
- AVG: average of the aggregated records
- MAX: maximum of the aggregated records
- MIN: minimum of the aggregated records
- GROUP_CONCAT: concatenates all aggregated records together, separated by a ","
- TOTAL: like SUM, but returns 0 instead of NULL when all aggregated records are NULL

For details see chapter 2 of the reading or
https://www.sqlite.org/lang_aggfunc.html

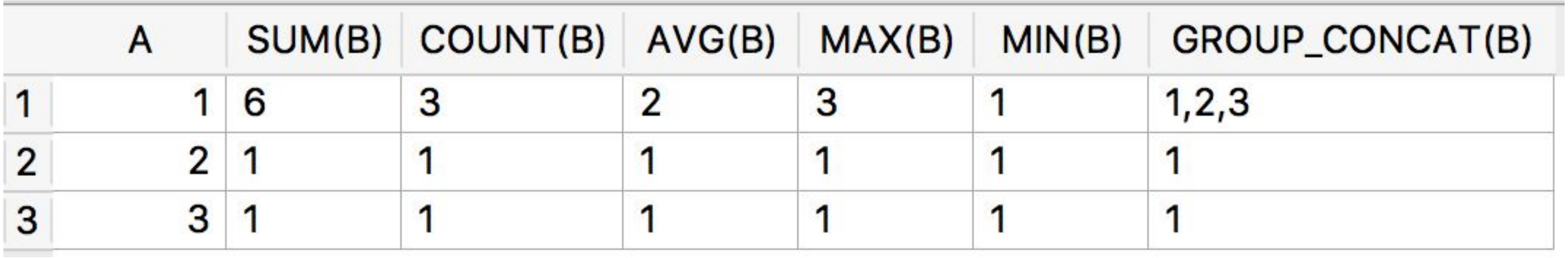
Example

```
SELECT  A,
        SUM(B),
        COUNT(B),
        AVG(B),
        MAX(B),
        MIN(B),
        GROUP_CONCAT(B)
FROM T
GROUP BY A
```

Table T



	A	B
1	1	1
2	1	2
3	1	3
4	2	1
5	3	1



	A	SUM(B)	COUNT(B)	AVG(B)	MAX(B)	MIN(B)	GROUP_CONCAT(B)
1	1	6	3	2	3	1	1,2,3
2	2	1	1	1	1	1	1
3	3	1	1	1	1	1	1

Here are some more details

The difference between SUM(X) and TOTAL(X) is this:
If all records are NULL, SUM returns NULL,
while TOTAL returns 0.

AVG, MIN, MAX, SUM, GROUP_CONCAT all return NULL if all aggregated records are NULL

COUNT(X) counts the records where X is not NULL
COUNT(*) counts all records

GROUP_CONCAT(X,Y) returns records concatenated with the separator in Y instead of ","

See the reading or https://www.sqlite.org/lang_aggfunc.html

Examples: GROUP BY details

- compare COUNT field

```
SELECT CustomerID, COUNT(ShippedDate) AS Count
FROM Orders
GROUP BY CustomerID ORDER BY Count DESC
```

- vs COUNT *

```
SELECT CustomerID, COUNT(*) AS Count
FROM Orders
GROUP BY CustomerID ORDER BY Count DESC
```

Examples: GROUP BY details

- compare TOTAL

```
SELECT Id, TOTAL(ShippedDate) AS Count
FROM Orders
GROUP BY Id ORDER BY Count ASC
```

- vs SUM

```
SELECT Id, SUM(ShippedDate) AS Count
FROM Orders
GROUP BY Id ORDER BY Count ASC
```

You can group by more than one field

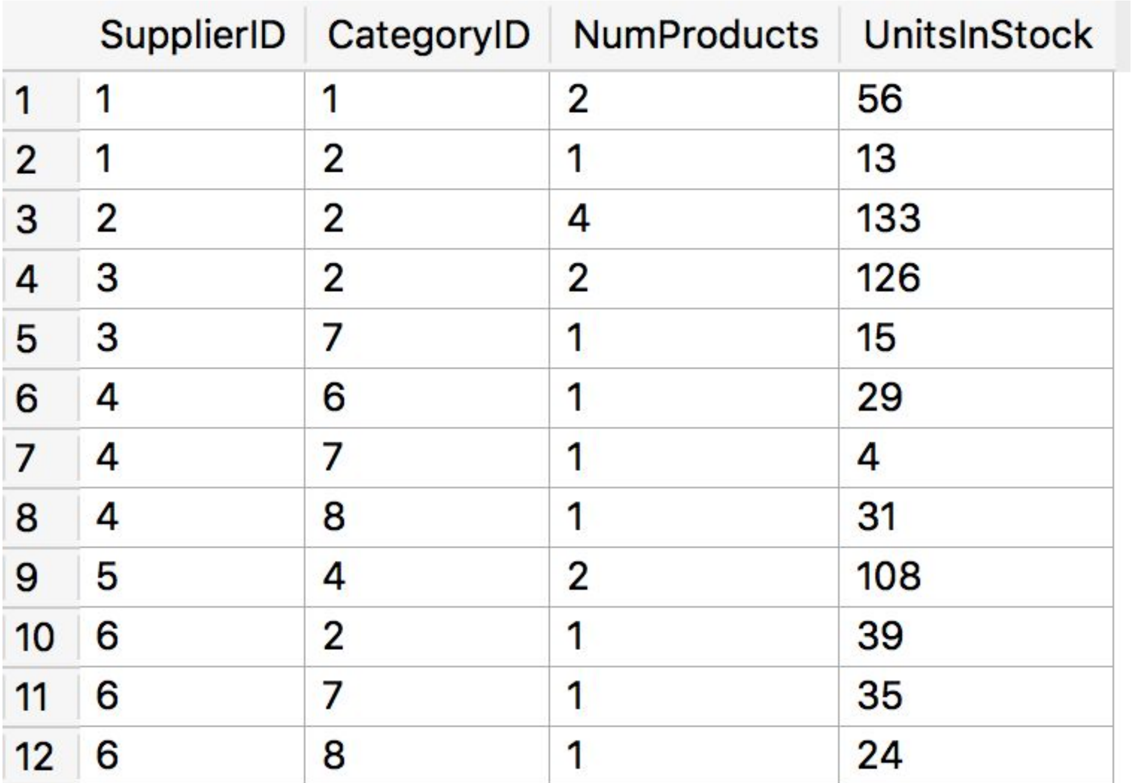
```
SELECT A, B, SUM(C) FROM T GROUP BY A, B
```

```
For each unique value of A in the table:
    For each unique value of B in the table:
        Finds all records with these values for A and B
        Compute the sum of field C for those records
```

You can also group by 3 fields, 4 fields, 5 fields, ...

Example

```
SELECT SupplierID, CategoryID, COUNT(*) AS NumProducts,
          SUM(UnitsInStock) AS UnitsInStock
FROM Products
GROUP BY SupplierID, CategoryID
```



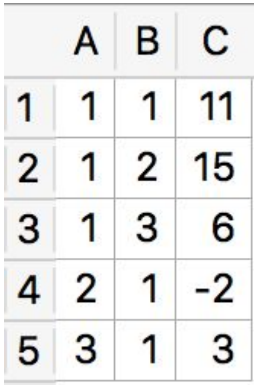
	SupplierID	CategoryID	NumProducts	UnitsInStock
1	1	1	2	56
2	1	2	1	13
3	2	2	4	133
4	3	2	2	126
5	3	7	1	15
6	4	6	1	29
7	4	7	1	4
8	4	8	1	31
9	5	4	2	108
10	6	2	1	39
11	6	7	1	35
12	6	8	1	24

You can group by calculated fields

These queries all produce the same records

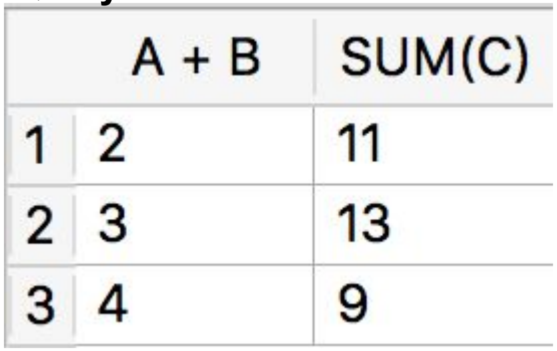
```
SELECT A+B, SUM(C) FROM T GROUP BY A+B
SELECT A+B AS AB, SUM(C) FROM T GROUP BY AB
SELECT A+B, SUM(C) FROM T GROUP BY 1
```

Table T



	A	B	C
1	1	1	11
2	1	2	15
3	1	3	6
4	2	1	-2
5	3	1	3

Query Result

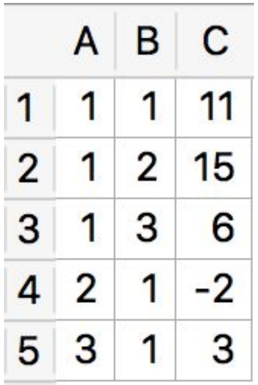


	A + B	SUM(C)
1	2	11
2	3	13
3	4	9

Which of these queries could have produced the screenshot below?

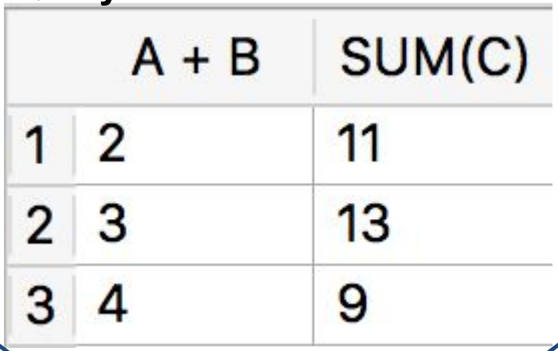
(a) SELECT A+B, SUM(C) FROM T GROUP BY A+B
(b) SELECT A+B AS AB, SUM(C) FROM T GROUP BY AB
(c) SELECT A+B, SUM(C) FROM T GROUP BY 1
(d) (a) or (b)
(e) (a) or (c)

Table T



	A	B	C
1	1	1	11
2	1	2	15
3	1	3	6
4	2	1	-2
5	3	1	3

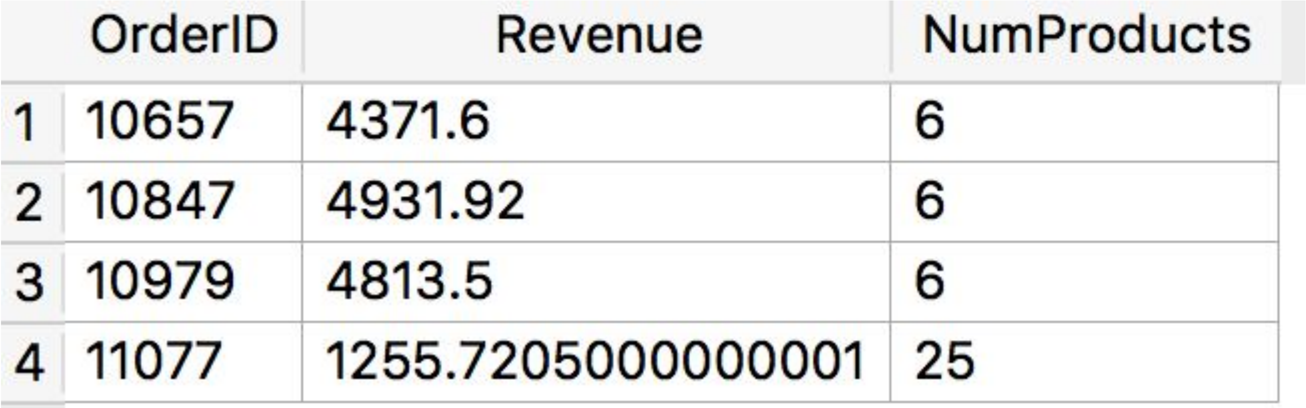
Query Result



	A + B	SUM(C)
1	2	11
2	3	13
3	4	9

You can filter records in a GROUP BY with HAVING

```
SELECT  OrderID,
        SUM(UnitPrice*Quantity*(1-Discount)) AS Revenue,
        COUNT(*) AS NumProducts
FROM OrderDetail
GROUP BY OrderID
HAVING COUNT(*)>5
```



	OrderID	Revenue	NumProducts
1	10657	4371.6	6
2	10847	4931.92	6
3	10979	4813.5	6
4	11077	1255.7205000000001	25

This is the same as creating a view and then filtering the view with WHERE

1. Create a view Q01 with the query:

```
SELECT  OrderID,
        SUM(UnitPrice*Quantity*(1-Discount)) AS Revenue,
        COUNT(*) AS NumProducts
FROM OrderDetail
GROUP BY OrderID
```

2. Run this query:

```
SELECT * FROM Q01 WHERE NumProducts>5
```

GROUP BY does not guarantee the order in which results are returned

In our example above, the results happened to be returned in order of OrderID.

That was just luck.

(More precisely, SQLite decided it was faster to return it that way, because of how the data is stored internally)

If you need a particular order, add an ORDER BY:

```
SELECT   OrderID,
         SUM(UnitPrice*Quantity*(1-Discount)) AS
      Revenue
FROM OrderDetail
GROUP BY OrderID
ORDER BY OrderID
```

SELECT statements without an ORDER BY do not guarantee the order in which results are returned

If you need a particular order, add an ORDER BY

Next lecture: JOIN