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.

<table>
<thead>
<tr>
<th>OrderID</th>
<th>Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>10248</td>
<td>168</td>
</tr>
<tr>
<td>10248</td>
<td>98</td>
</tr>
<tr>
<td>10248</td>
<td>174</td>
</tr>
<tr>
<td>10249</td>
<td>167.4</td>
</tr>
<tr>
<td>10249</td>
<td>1696</td>
</tr>
<tr>
<td>10250</td>
<td>77</td>
</tr>
<tr>
<td>10250</td>
<td>1261.399999999999999</td>
</tr>
<tr>
<td>10250</td>
<td>214.2</td>
</tr>
<tr>
<td>10251</td>
<td>95.76</td>
</tr>
<tr>
<td>10251</td>
<td>222.299999999999998</td>
</tr>
<tr>
<td>10251</td>
<td>336</td>
</tr>
<tr>
<td>10252</td>
<td>2462.4</td>
</tr>
<tr>
<td>10252</td>
<td>475</td>
</tr>
</tbody>
</table>
SQL can aggregate records by OrderID and sum them

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

Result from query on the previous slide
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
What records does this query produce?

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

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

(a) A | SUM(B)  
---|-------
1 | 1  
1 | 2  
1 | 3  
2 | 1  
3 | 1

(b) A | SUM(B)  
---|-------
1 | 6  
2 | 1  
3 | 1

(c) A | SUM(B)  
---|-------
6 | 1  
1 | 2  
1 | 3

(d) A | SUM(B)  
---|-------
8 | 8

(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>
<thead>
<tr>
<th>A</th>
<th>SUM(B)</th>
<th>COUNT(B)</th>
<th>AVG(B)</th>
<th>MAX(B)</th>
<th>MIN(B)</th>
<th>GROUP_CONCAT(B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1,2,3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>SupplierID</th>
<th>CategoryID</th>
<th>NumProducts</th>
<th>UnitsInStock</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>56</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
<td>133</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>126</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>1</td>
<td>29</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>1</td>
<td>31</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>2</td>
<td>108</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>1</td>
<td>39</td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>1</td>
<td>35</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>1</td>
<td>24</td>
</tr>
</tbody>
</table>
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
```
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)
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
```
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