This reading describes two techniques you can use to save time and make your SQL queries more concise when you’re using the **GROUP BY**clause.

For example, consider this query from the “Tips for Applying Grouping and Aggregation” video:

**SELECT MIN(dep\_time), MAX(dep\_time), COUNT(\*)**

**​   FROM flights**

**​   GROUP BY CASE WHEN dep\_time IS NULL then 'missing'**

**​    ​    ​    ​   WHEN dep\_time < 500 then 'night'**

**​    ​    ​    ​   WHEN dep\_time < 1200 THEN 'morning'**

**​    ​    ​    ​   WHEN dep\_time < 1700 THEN 'afternoon'**

**​    ​    ​    ​   WHEN dep\_time < 2200 THEN 'evening'**

**​    ​    ​    ​   ELSE 'night'**

**​    ​   END;**

Note that this query doesn’t actually include the grouping column in the output:

**Results**

| **min(dep\_time)** | **max(dep\_time)** | **count(\*)** |
| --- | --- | --- |
| 1200 | 1659 | 18366410 |
| 500 | 1159 | 24513240 |
| NULL | NULL | 961944 |
| 1 | 2400 | 2067458 |
| 1700 | 2159 | 15483770 |

Instead it uses **MIN(dep\_time)**and **MAX(dep\_time)**as a way to indicate which of these time bins reach row in the result set represents. This results in a curious row that appears to encompass all values from **1**to **2400**; this is actually the group defined by **WHEN dep\_time < 500 then 'night'**(including values from **0**to **499**) and the **ELSE 'night'**(including values from **2200**to **2400**).

To include the grouping column in the output with Hive and some other SQL engines, you would have to do this:

**SELECT CASE WHEN dep\_time IS NULL then 'missing'**

**​    ​    ​   WHEN dep\_time < 500 then 'night'**

**​    ​    ​   WHEN dep\_time < 1200 THEN 'morning'**

**​    ​    ​   WHEN dep\_time < 1700 THEN 'afternoon'**

**​    ​    ​   WHEN dep\_time < 2200 THEN 'evening'**

**​    ​    ​   ELSE 'night'**

**​    ​   END AS dep\_time\_category,**

**​    ​   COUNT(\*)**

**​   FROM flights**

**​   GROUP BY CASE WHEN dep\_time IS NULL then 'missing'**

**​    ​    ​   WHEN dep\_time < 500 then 'night'**

**​    ​    ​   WHEN dep\_time < 1200 THEN 'morning'**

**​    ​    ​   WHEN dep\_time < 1700 THEN 'afternoon'**

**​    ​    ​   WHEN dep\_time < 2200 THEN 'evening'**

**​    ​    ​   ELSE 'night'**

**​    ​   END;**

**Results**

| **dep\_time\_category** | **count(\*)** |
| --- | --- |
| afternoon | 18366410 |
| morning | 24513240 |
| missing | 961944 |
| night | 2067458 |
| evening | 15483770 |

That gives a result that's more easily understood, but it's a long, repetitive query!

Using an Alias in the **SELECT**List

With Impala, MySQL, and PostgreSQL, you can use an alias in the **SELECT**list and then refer to it in the **GROUP BY**clause. That is, you can use this query instead:

**SELECT CASE WHEN dep\_time IS NULL then 'missing'**

**​    ​    ​   WHEN dep\_time < 500 then 'night'**

**​    ​    ​   WHEN dep\_time < 1200 THEN 'morning'**

**​    ​    ​   WHEN dep\_time < 1700 THEN 'afternoon'**

**​    ​    ​   WHEN dep\_time < 2200 THEN 'evening'**

**​    ​    ​   ELSE 'night'**

**​    ​   END AS dep\_time\_category,**

**​    ​   COUNT(\*)**

**​   FROM flights**

**​   GROUP BY dep\_time\_category;**

This produces the same results, but in a more concise query.

Using Positional References

Another way to do this with Impala, MySQL, PostgreSQL, and newer versions of Hive (but not in older versions of Hive and not in some other SQL engines) is to use an integer (1, 2, and so on) as the grouping expression, and the engine will use the corresponding column in the **SELECT**list as the grouping column. If you use **GROUP BY 3**, then the third column you specify in your **SELECT**list will be the grouping column.

This means you could also use this query to get the same results for the departure time category:

**SELECT CASE WHEN dep\_time IS NULL then 'missing'**

**​    ​    ​   WHEN dep\_time < 500 then 'night'**

**​    ​    ​   WHEN dep\_time < 1200 THEN 'morning'**

**​    ​    ​   WHEN dep\_time < 1700 THEN 'afternoon'**

**​    ​    ​   WHEN dep\_time < 2200 THEN 'evening'**

**​    ​    ​   ELSE 'night'**

**​    ​   END AS dep\_time\_category,**

**​    ​   COUNT(\*)**

**​   FROM flights**

**​   GROUP BY 1;**

Since **dep\_time\_category**is the first column in the **SELECT**list, **GROUP BY 1**directs the SQL engine to group by that column.

*NOTE:*In general, this shortcut method is less preferable, because it's harder to see what your query does, and it could cause trouble if you changed your **SELECT**list but forgot to change your **ORDER BY**clause.

Another Example

The tables in the **fly**database are not available to the MySQL and PostgreSQL engines in the VM. If you want to test this on the VM using either of those databases, you can try these queries:

Using an alias:

**SELECT CASE**

**​    ​    ​   WHEN price <= 10 THEN 'inexpensive'**

**​    ​    ​   WHEN price > 10 THEN 'expensive'**

**​    ​    ​   ELSE 'unknown'**

**​    ​   END AS price\_category,**

**​    ​   COUNT(\*)**

**​   FROM inventory**

**​   GROUP BY price\_category;**

Using positional reference:

**SELECT CASE**

**​    ​    ​   WHEN price <= 10 THEN 'inexpensive'**

**​    ​    ​   WHEN price > 10 THEN 'expensive'**

**​    ​    ​   ELSE 'unknown'**

**​    ​   END AS price\_category,**

**​    ​   COUNT(\*)**

**​   FROM inventory**

**​   GROUP BY 1;**

Care must be taken when grouping. It's possible to produce misleading results, or even results that seem contradictory.

In the **fly.flights**table, if you compare average on-time performance, **AVG(arr\_delay)**, over all flights for the carriers Virgin America (carrier code **VX**) and SkyWest Airlines Inc. (carrier code **OO**), then SkyWest has a better average delay (approximately 5.7 minutes) than Virgin (approximately 6.5 minutes). You might conclude, then, that Virgin is worse than SkyWest in terms of delays, and when given a choice for a particular trip between two cities, choose SkyWest.

However, Virgin would actually be a slightly better choice in that case (and if arrival delay is your only criterion)! If you limit the data to the airports where both airlines have flights, then Virgin looks slightly better than SkyWest (9.5 minutes for Virgin and 9.7 minutes for SkyWest). It might not be a problem with Virgin being worse than SkyWest, then, in terms of delays. Instead, the problem could be with the airports where they operate. The airports where Virgin operates overall have worse delays than the airports where SkyWest operates, so Virgin's average on-time performance over *all*flights looks worse than SkyWest.

(Note: The queries for these comparisons require some techniques you haven't learned yet, but they are included at the end of this reading in case you want to try them.)

In this case, the airports is a *confounding variable*—an underlying variable that affects each of the other variables that you are examining. The airports themselves can be a source of delay (San Francisco often has delays on account of fog, for example), and the carriers work with different airports. This underlying variable makes a difference, so a good comparison needs to accommodate that variable.

Another example is Simpson's Paradox, in which grouping in one way can provide one conclusion for every single group, but when taken as a whole, the *opposite*conclusion is reached. This is often because of a significant difference in sample size for the groups. For example, a study of kidney stone treatment found that while one treatment appeared to be more effective for both small stones and for large stones, when you looked at all the cases together, the other treatment appeared to be more effective. (See the Wikipedia page, Simpson's paradox, for this and other examples.  “Simpson's Paradox: How to Prove Opposite Arguments with the Same Dataset”  also explains this phenomenon well.)

|  | **Treatment A** | **Treatment B** |
| --- | --- | --- |
| **Small stones** | *Group 1***93% (81/87)** | *Group 2*87% (234/270) |
| **Large stones** | *Group 3***73% (192/263)** | *Group 4* 69% (55/80) |
| **Both** | 78% (273/350) | **83% (289/350)** |

Notice that treatment A was provided about three times as much for large stones as for small stones, while treatment B was provided about three times as much for small stones as for large stones. The severity (size) of the stones is the confounding variable.

To avoid making conclusions from misleading data, you might:

* Use different levels of aggregation, including the highest aggregation and no aggregation at all, if possible, when looking at the data;
* Try different ways to group your data, to be sure your choice of groups isn't causing an effect that disappears or reverses for different choices; and
* Include counts in your results so you can see when one group has a significantly different number of contributions than others.

Queries Used to Compare Carriers

These queries use some techniques you haven't learned yet, but they are included here in case you want to try them.

Comparing Virgin to SkyWest, all flights:

**SELECT carrier, AVG(arr\_delay), COUNT(arr\_delay)**

 ​   **FROM flights WHERE carrier='VX' OR carrier='OO'**

 ​   **GROUP BY carrier ORDER BY avg(arr\_delay);**

| **carrier** | **avg(arr\_delay)** | **count(arr\_delay)** |
| --- | --- | --- |
| OO | 5.685446716780021 | 5926697 |
| VX | 6.484657383617123 | 367408 |

 Comparing Virgin to SkyWest, identical origins and destinations:

**SELECT f.carrier, avg(f.arr\_delay), count(f.arr\_delay)**

 ​   **FROM flights f**

 ​   **JOIN**

 ​    ​   **(SELECT DISTINCT origin, dest FROM flights WHERE carrier='VX') vx**

 ​    ​    ​   **ON (f.origin=vx.origin AND f.dest=vx.dest)**

 ​   **JOIN**

 ​    ​   **(SELECT DISTINCT origin, dest FROM flights WHERE carrier='OO') oo**

 ​    ​    ​   **ON (f.origin=oo.origin AND f.dest=oo.dest)**

 ​   **WHERE carrier='VX' OR carrier='OO'**

 ​   **GROUP BY f.carrier ORDER BY avg(f.arr\_delay);**

Note that this query uses the **JOIN**keyword to combine tables; you'll learn about this in Week 6 of this course. It also uses *subqueries*to isolate the origin/destination pairs for each carrier; you'll learn about subqueries if you continue to the fourth course in this specialization.

| **carrier** | **avg(f.arr\_delay)** | **count(f.arr\_delay)** |
| --- | --- | --- |
| OO | 9.72568710815216 | 231914 |
| VX | 9.546630527151848 | 180043 |