



In this lab, you will again use the PHPMyAdmin web interface to your database. You will work with the tables from the airline database you installed for lab sheet 8 and you will modify them during this lab. You will learn the difference between grouping data sets by attributes and ordering them, and you will get to meet a number of functions, both aggregate and non-aggregate.

Note again that these lab sheets are not assessed, but they may be discussed in tutorials and their content is examinable!

Estimated time to complete this lab: 30 minutes.

A) Sorting and ordering

Remember this query from lab sheet 8? It computed the return flight distance for each aircraft in our fleet:

```
SELECT aircraft_type, registration, 2*sum(distance)
FROM Aircraft
LEFT JOIN Route
ON Route.aircraft_id = Aircraft.aircraft_id
GROUP BY registration;
```

Remember that this introduced **sum()** as our first aggregate function and an example for **GROUP BY**. What actually happens here is that the RDBMS, in this order:

1. Carries out the join
2. Sorts the join according to aircraft registration
3. Computes the sum over each group and multiplies it by 2
4. Assembles the result record set by taking the first `aircraft_type` and `registration` values from each group and putting them into the same result set record.

It's still important to remember though that these two values are actual field values – we just know that they will be the same across the group. If you ignore this, you get gibberish. Make a note of the mileages and then extend the query just a little:

```
SELECT route_description, aircraft_type, registration, 2*sum(distance)
FROM Aircraft
LEFT JOIN Route
ON Route.aircraft_id = Aircraft.aircraft_id
GROUP BY registration;
```

Notice something? MySQL has now grabbed the first route description at hand, but the route description is just *one* of generally *several* in the group, and so has no real relationship to the distance sum. So, when you use **GROUP BY**, ensure that the list of fields that you retrieve in the **SELECT** statement will have identical values across the group, or else you get nonsense.

So that query did some sorting (according to registration), but what if we want the final result sorted by aircraft type? Then we can't escape the **ORDER BY** clause, which is the clause you always use if all you need is having your result set sorted.

```
SELECT aircraft_type, registration, 2*sum(distance)
FROM Aircraft
LEFT JOIN Route
ON Route.aircraft_id = Aircraft.aircraft_id
GROUP BY registration
ORDER BY aircraft_type;
```

You don't need to sort by a table field – you can also sort by an aggregate function:

```
SELECT aircraft_type, registration, 2*sum(distance)
FROM Aircraft
LEFT JOIN Route
ON Route.aircraft_id = Aircraft.aircraft_id
GROUP BY registration
ORDER BY sum(distance);
```

or by more than one criterion:

```
SELECT aircraft_type, registration, 2*sum(distance)
FROM Aircraft
LEFT JOIN Route
ON Route.aircraft_id = Aircraft.aircraft_id
GROUP BY registration
ORDER BY aircraft_type, sum(distance);
```

For an individual ordering criterion, we can specify the direction of the ordering. The default is ASC (“ascending”), meaning that lower values come first. With DESC (“descending”), the higher values are given the priority. For example:

```
SELECT aircraft_type, registration, 2*sum(distance)
FROM Aircraft
LEFT JOIN Route
ON Route.aircraft_id = Aircraft.aircraft_id
GROUP BY registration
ORDER BY aircraft_type, sum(distance) DESC;
```

lists the aircraft of the same type but with higher mileage first.

B) Useful functions

Going back to the query above, note that it returns a NULL field for the aircraft ZK-DUCK that hasn’t really been assigned to any duties yet. Being computer folk, we know a NULL when we see one and we know what it means (or, rather, doesn’t mean). But the airline operations manager who will see your report would probably get quite upset if we were to print out a NULL here. A function we can use to have something more meaningful here is IFNULL():

```
SELECT aircraft_type, registration,
       IFNULL(2*sum(distance), 'aircraft unassigned')
FROM Aircraft
LEFT JOIN Route
ON Route.aircraft_id = Aircraft.aircraft_id
GROUP BY registration
ORDER BY aircraft_type, sum(distance) DESC;
```

Be aware though that this can cause you difficulties with data types in client applications: If your client expected a number from this query, then getting a string “aircraft unassigned” back may cause an error.

Another thing that gets our poor manager upset are column headers with strange names, so let’s fix that as an aside while we’re at it:

```
SELECT aircraft_type as `aircraft type`, registration,
       IFNULL(2*sum(distance), 'aircraft unassigned') as `km per week`
FROM Aircraft
LEFT JOIN Route
ON Route.aircraft_id = Aircraft.aircraft_id
GROUP BY registration
ORDER BY aircraft_type, sum(distance) DESC;
```

This looks a lot more civilised now, doesn’t it. One smart application of this field aliasing is to adapt field names from legacy databases to property names of objects in your application, by the way.

Another set of useful functions are the date and time functions. The query:

```
SELECT CURDATE();
```

gives you the current date, for example. Let’s add some datetime data to our **Aircraft** table. You can apply the following changes by importing the script *airline_inspection.sql* from the Lecture page on the course website:

```
ALTER TABLE Aircraft ADD inspection_date DATETIME AFTER registration;
UPDATE Aircraft SET inspection_date = '2011-04-25' WHERE registration = 'ZK-BLOB';
UPDATE Aircraft SET inspection_date = '2011-05-03' WHERE registration = 'ZK-DUCK';
UPDATE Aircraft SET inspection_date = '2011-11-20' WHERE registration = 'ZK-BIGG';
```

```
UPDATE Aircraft SET inspection_date = '2011-10-18' WHERE registration = 'ZK-WHIZ';
UPDATE Aircraft SET inspection_date = '2011-06-23' WHERE registration = 'ZK-HUGE';
UPDATE Aircraft SET inspection_date = '2011-09-12' WHERE registration = 'ZK-GIGA';
```

Once done, try the following query:

```
SELECT registration, DATEDIFF(inspection_date, CURDATE())
      FROM Aircraft;
```

This gives you the number of days between today and the date that the next inspection for the respective aircraft is due. The DATEDIFF() function computes the difference between two dates in days, the first of which is supplied by **inspection_date** and the second of which is supplied by CURDATE(). As you can see, we can use function return values as function parameters in SQL, too.

C) Average days to inspection

If we want to know the average of a field (or a field in a group), we can use the AVG() function much like we use the SUM() function or the COUNT() function (in fact, it's just sum divided by count anyway).

Exercise: Construct a query that computes the average number of days to the next inspection for the different aircraft types in our fleet.