



In this lab, you will again use the PHPMyAdmin web interface to your database. You will need to install two database tables via the IMPORT tab. The exercises will then take you through a variety of join techniques.

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) Airline operations

To start this exercise, import the file *airline.sql* into your database via the Import tab of the PHPMyAdmin web interface. It creates two tables, **Aircraft** and **Route** as follows:

```
CREATE TABLE Aircraft (  
    aircraft_id INT UNSIGNED NOT NULL AUTO_INCREMENT,  
    aircraft_type varchar(4),  
    registration varchar(10),  
    PRIMARY KEY (aircraft_id)  
) ENGINE=InnoDB;
```

```
CREATE TABLE Route (  
    route_id INT UNSIGNED NOT NULL AUTO_INCREMENT,  
    route_description varchar(100),  
    aircraft_id INT UNSIGNED,  
    distance INT UNSIGNED,  
    PRIMARY KEY (route_id),  
    INDEX ix_route_aircraft_id (aircraft_id)  
) ENGINE=InnoDB;
```

The script in the file also inserts a few records into each table. The **Aircraft** table stores details on the airline's fleet. Each record corresponds to one individual aircraft and stores its type and its registration number (also known as the plane's call sign), as well as an integer primary key that auto-increments. The **Route** table stores information on each route that the airline services. Apart from an auto-incrementing primary integer key, it stores the route description and the id of the aircraft that is used to service the route. The **aircraft\_id** field in the table references the **aircraft\_id** from the **Aircraft** table, but as we're not enforcing a foreign key relationship here, we can have values in this field that do not occur in its counterpart in the **Aircraft** table – in fact, we do: The Auckland-Perth, Auckland-Tokyo and Auckland-Honolulu routes all have an aircraft id that does not belong to any aircraft in the **Aircraft** table. The **Route** table also stores the one-way distance that the route covers (in km). Do a select on both tables to familiarise yourself with the contents.

### B) Show me all routes that have aircraft assigned to them

This is an inner join. Show me the route, the distance, and the type and registration number of the aircraft used. This can be done in the following ways:

```
SELECT route_description, distance, aircraft_type, registration  
FROM Route, Aircraft  
WHERE Route.aircraft_id = Aircraft.aircraft_id;
```

or using inner JOIN syntax:

```
SELECT route_description, distance, aircraft_type, registration  
FROM Route  
JOIN Aircraft  
ON Route.aircraft_id = Aircraft.aircraft_id;
```

or

```
SELECT route_description, distance, aircraft_type, registration  
FROM Aircraft
```

```
JOIN Route
ON Route.aircraft_id = Aircraft.aircraft_id;
```

Try all three and satisfy yourself that they all give the same record set. Note that in some circumstances, the order of the records may vary – remember in an RDMBS there is no guarantee in which order they come out unless you request them to be ordered.

Note that the Auckland-Perth, Auckland-Tokyo and Auckland-Honolulu routes don't show up, because they don't have any existing aircraft assigned to them. Note also that our B737 ZK-DUCK never gets to take off, because it's not been assigned to any route. That's your inner join for you: no guarantees that records from either table have a counterpart in the other.

### **B) Show me all routes and any aircraft assigned to them**

This is a left join for you (if the **Route** table is in the FROM clause):

```
SELECT route_description, distance, aircraft_type, registration
FROM Route
LEFT JOIN Aircraft
ON Route.aircraft_id = Aircraft.aircraft_id;
```

Note that now, you'll see all routes but some will have a NULL for **aircraft\_type** and **registration**.

### **C) Show me all aircraft and any routes assigned to them**

Easy peasy: just swap the tables around:

```
SELECT route_description, distance, aircraft_type, registration
FROM Aircraft
LEFT JOIN Route
ON Route.aircraft_id = Aircraft.aircraft_id;
```

Now we get to see all planes but there's a "route" that's all NULL – that of the flightless ZK-DUCK.

Alternatively, we could do a right join:

```
SELECT route_description, distance, aircraft_type, registration
FROM Route
RIGHT JOIN Aircraft
ON Route.aircraft_id = Aircraft.aircraft_id;
```

which gives the same result.

### **D) Does this work the other round for B), too?**

Sure does. Try this:

```
SELECT route_description, distance, aircraft_type, registration
FROM Aircraft
RIGHT JOIN Route
ON Route.aircraft_id = Aircraft.aircraft_id;
```

Simple rule: in a LEFT JOIN, the table in the FROM clause gets to show all its records in the result set, in a RIGHT JOIN, the table after the JOIN keyword gets to show all its records.

### **E) Getting the big picture**

Now what if we want to see all routes and all aircraft, no matter whether the route has an aircraft assigned or the aircraft is assigned to a route? You'll need to do an outer join for this. As MySQL does not offer outer joins, we need to use a gambler's trick: get a left join as under B) and one of the joins under C) and combine them with a UNION, e.g., like so:

```
SELECT route_description, distance, aircraft_type, registration
FROM Route
LEFT JOIN Aircraft
ON Route.aircraft_id = Aircraft.aircraft_id
UNION
```

```

SELECT route_description, distance, aircraft_type, registration
FROM Route
RIGHT JOIN Aircraft
ON Route.aircraft_id = Aircraft.aircraft_id;

```

Note that all we've done here is copied the two statements across, removed the semicolon after the first, and stuck the word UNION in instead. Easy. Note that records that occur both in the result set of the first query and in the result set of the second query only turn up once. Note also that when we say "records" here, we mean the records in the result set(s), not these in the original table. This can easily be seen by removing the route description and distance fields from the two subqueries on the UNION above – now each plane only turns up once!

### **F) Getting it all sorted**

One of the not so nice things about record sets is that they don't come in any particular order, unless of course we ask. The easiest thing we could do above is add an **ORDER BY route\_description** to any of the queries and we'd get everything sorted in alphabetical order. Try this out for a couple of queries there.

Another useful clause is the GROUP BY clause. It works like the ORDER BY clause and will give the same result if you substitute it in the experiments you just conducted. However, it will only work with fields that are actually on the SELECT list of fields (ORDER BY will happily work with any field that could be on the SELECT list). So why would we want to use it if ORDER BY seems more flexible?

The answer: because it also works with *aggregate functions*. Say we wanted to find out how many km each of our aircraft flies in a week (there and back, of course, which is twice the distance), assuming that all of these are weekly flights. Then we can do this:

```

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

```

This shows that our A380 ZK-GIGA is our main workhorse with 45200 km a week. In this case, **sum()** is one of those famous aggregate functions (others are **max()**, **min()**, etc.) which are applied to each group of records – in this case, each group formed by a common value in the **registration** field.

Exercise: Change the query above to find out how many weekly km our respective fleets of different aircraft types complete. Note: You'll want to remove the **registration** field from the SELECT list above – but leave it in at first to see what happens, and why this could be misleading!