



In this lab, you will again connect to the MySQL server with your database via the faculty's login server. If you have not done so yet, please complete the Lecture01 Lab first. There you would have set up a table in the database which you populated with some data on cities. You will need this setup for this lab. In this lab, you will practise retrieving data from your database with SELECT and you will also add a primary key to your table.

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: 30minutes.

A) Connecting to the Faculty of Science login server and the MySQL server

See the Lecture 01 Lab. For the next sections, we will assume that you are already logged into the MySQL server.

B) Finding a particular city

Say that Auckland is one of the cities in your **city** table. Then you can look up its record as follows:

```
SELECT * FROM city WHERE name = 'Auckland';
```

C) Getting a key

However, if you input Christchurch in this query, then this could be a little ambiguous: There is not just the Christchurch in the South Island, but also the Christchurch on the South coast of England. Which one do we mean? Clearly, the name of a city isn't always unique: There's a Paris in France and one in Texas, and there's a Melbourne in Australia and one in Florida... the list goes on. What we need is a unique identifier – a unique key.

OK, you might say, let's use a key as a combination of name and country. So then we have two Palmerstons in NZ, two Frankfurts in Germany, so this isn't ideal either. We need a unique primary key, a number that identifies each city regardless of name, location, or size. Ideally, we don't want to have to worry about specifying that number either – we want it to be assigned automatically each time we insert a new city record.

So, first of all, we need a new field. Let's call it **cityId**:

```
ALTER TABLE city ADD cityId int FIRST;
```

This adds the new field at the beginning (FIRST) of the row in the table. The next step would be to declare this field as a primary key, but there is a problem: If we try

```
ALTER TABLE city ADD PRIMARY KEY(cityId);
```

we get an error. Why? Because the primary key needs to be unique, and since our **city** table already has several records in it, the **cityId** for all of these was set to NULL when we added the field. So all the records now have the same value and it's anything but unique. Do a

```
SELECT * FROM city;
```

to verify this. Bummer! Do not despair – some MySQL wizardry will help: a user-defined variable and a subquery will come to the rescue. First, do this:

```
SET @x = 0;
```

This creates a user-defined variable **@x** and sets it to 0. Easy, right? Now the real magic starts here:

```
UPDATE city SET cityId = (SELECT @x := @x + 1);
```

Note that this UPDATE doesn't have a WHERE clause, so it affects all records in the table. It sets the **cityId** field of each record to the value of **@x**. Each time it does this, it also increments **@x**, providing the value for the next row. Smart! Do another

SELECT on the table to verify that each row now has a unique **cityId**. Note that this wizardry doesn't work on all versions of SQL or even MySQL – they must support subqueries at the very least.

Now we're ready to get the primary key organised:

```
ALTER TABLE city ADD PRIMARY KEY(cityId);
```

In the final step, we need to make the field auto-incrementing:

```
ALTER TABLE city MODIFY cityId int auto_increment;
```

Done!

D) Adding more cities

Now add another record to the **city** table, using the same INSERT syntax as before. Note that you do not need (and in fact should not) specify the **cityId** value explicitly – just pass in NULL. After the insert, you can retrieve the latest **cityId** with:

```
SELECT LAST_INSERT_ID();
```

This is a MySQL function that is intended specifically for this purpose.

Do another SELECT on the table to verify that the **cityId** returned is in fact the one that **LAST_INSERT_ID()** returned.

E) Retrieving individual records with various WHERE clauses

We can now retrieve a specific city record like this:

```
SELECT * FROM city WHERE cityId = 3;
```

However, we can also use more complex WHERE clauses. As an exercise, try:

- Retrieving all cities that have at least a certain population
- Retrieving all cities in a particular country
- Retrieving all cities in a particular country that have at least a certain population

Depending on the data in your table, all these queries may return more than one record.