Databases and the Web – Exercise 2

This exercise builds on last week's experience with Linux and MySQL to build a non-trivial database that we'll use in the future.

The purpose of exercise 2 is to practice/revise using SQL, become familiar with using phpMyAdmin and building MySQL database tables.

Task 0: Finish exercise 1

Ensure you know how to:

1. Edit/create a PHP document in Windows.
2. Upload a document to the StudentNet server.
3. Use a browser to view a document from the StudentNet server.
4. Tell the difference between client-side and server-side code.

Task 1: Import some data *manually* into a MySQL database

1) Open Firefox & load the "phpMyAdmin" page to manipulate your StudentNet database.

2) Using phpMyAdmin, create an InnoDB table called **`country`**. It has 15 columns and you must decide the most appropriate column type to use from either lecture 2's slides or the MySQL manual.

   i) The columns are as follows (*in this order*):

1) Code	• Always 3 letters & also the table's primary key.
2) Name	• At most 52 letters.
3) Continent	• Selected from the set 'Asia', 'Europe', 'North America', 'Africa', 'Oceania', 'Antarctica', 'South America'.
4) Region	• At most 26 letters.
5) SurfaceArea	• Floating point number with max 10 digits and 2 decimal placess.
6) IndepYear	• An integer year value, such as 1999, might be NULL.
7) Population	• An integer with max 11 digits
8) LifeExpectancy	• A floating point number with max 3 digits and 1 decimal place, could be nullable.
9) GNP	• A floating point number with max 10 digits and 2 decimal places. NULL? Consider, does Antarctica have a GNP?
10) GNPOld	
11) LocalName	• At most 45 letters.
12) GovernmentForm	
13) HeadOfState	• At most 60 letters, possibly NULL.
14) Capital	• An 11 digit integer. (Becomes a *foreign key* in Task 3). NULL?
15) Code2	• 2 characters

   ♦ Instead of point-and-click in phpMyAdmin you could build the necessary "CREATE" SQL command in the textbox or using an editor (then copy/paste).

ii) Make sure you verify the table structure in phpMyAdmin or with MySQL's `DESCRIBE country;`

3) Import data into **`country`** from this country.csv file using phpMyAdmin's "Import" tab.

- Glance through the data – CSV files are common formats although they lack "meta data" that explains what the columns *mean*.

4) Check it!

- If you enter "**`SELECT COUNT(*) FROM country;`**" you should get 239.
- "**`SELECT SUM(GNP) FROM country;`**" the total **`GNP`** is 29354907.9 (**`SUM`** is an aggregate function & there are many in MySQL.)

Congratulations! You have just imported one table from the Finnish world countries database <grin> (MySQL AB used to use it as their example database…)

5) Try some queries in the phpMyAdmin SQL tab:

i) List the names of the 18 countries that became independent in 1991.

ii) List the 5 countries in Southern Europe with population less than 1 000 000.

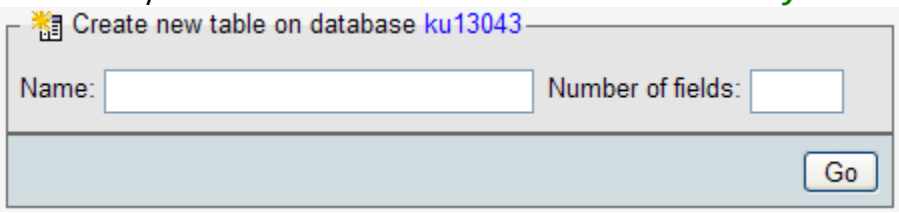
iii) What's the *continent* that contains the *country* with the maximum life expectancy? How about the minimum?

iv) What's the *continent* with the best *average* life expectancy?You could use an "aggregate function" with a GROUP BY clause in MySQL – look in the manual?

- Kudos to the first person to post the right answers to the StudySpace discussion forum <grin>

Task 2: Create and populate the `city` table

1) Go back to your database view and create a table called **`city`** with 5 fields



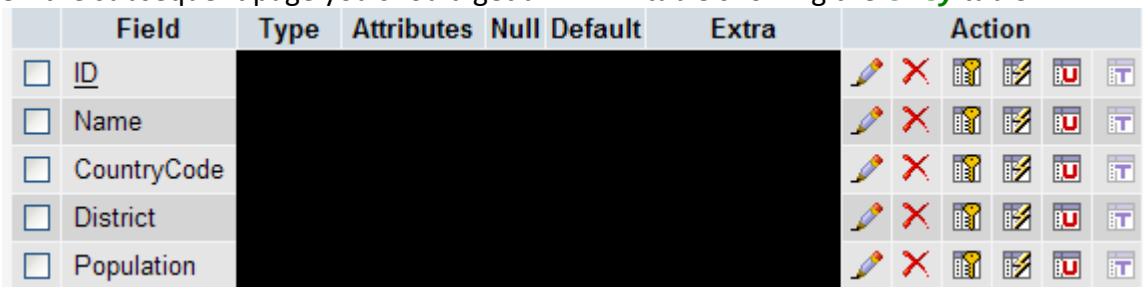
2) Its fields are

1) ID	• An integer field with at most 11 digits that is also the primary key.
2) Name	• 35 letters
3) CountryCode	• 3 letters, a foreign key that references country's code field.
4) District	• 20 letters.
5) Population	• An integer field with at most 11 digits.

Make sure you find the relevant drop-downs and buttons to assign each field's properties in phpMyAdmin…

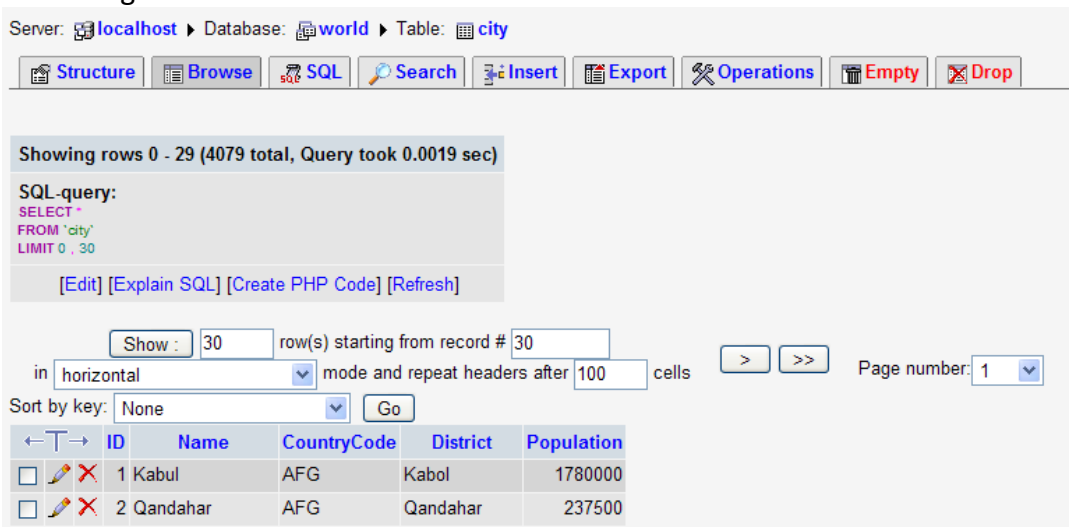
- By using InnoDB tables we can arrange for the foreign key to be correctly associated with the **`country`** table ☺

3) On the subsequent page you should get an HTML table showing the **`city`** table:



4) Repeat the import process (from earlier) using this city.csv file.

   i) What you should end up with is a table with 4079 rows whose "browse" page starts something like this:



Task 3: Create and populate the `countrylanguage` table

1) Final table! This time we want to create **`countrylanguage`** with these fields:

1) CountryCode	• 3 letters, a foreign key that references country's code field that is also part of the composite primary key.
2) Language	• 30 letters that is the other part of the composite primary key.
3) IsOfficial	• Boolean, simulated by a field that takes one of two values: "T" or "F" only.
4) Percentage	• A floating point number representing a percentage with 3 digits and 1 decimal place.

2) However there is a quick and easy way to do this in one step ;-) just import this "SQL dump" backup file:

http://staffnet.kingston.ac.uk/~ku13043/WebDB/ex/week02/countrylanguage.sql
(You should browse through the file to see the structure.)

3) Finally, ensure the referential integrity constraints are in place:

   i) There's a nice simple command to make any table InnoDB type, *e.g.*:

   - **`ALTER TABLE countrylanguage TYPE=InnoDB;`**

   (There's a straightforward way in the phpMyAdmin GUI too …)

   ii) If you have not already done-so, set-up the appropriate primary keys.

   - *E.g.* from the "Structure" view for country highlight the key and press the  button.

   - For a composite key, highlight the rows and press the  button beneath:

   iii) Look at the foreign keys – which table "owns" the key and which table "uses" a copy of the key? The parent should be allowed to DELETE or UPDATE a key field and have that deletion *cascade* into the child table, *not* the other way around and **not** in both directions either! (Circular constraints will prevent all UPDATE/INSERT/DELETE operations.)

   iv) From the structure tab in the relevant tables, open the "Relation view" and link the keys as appropriate.

   - If you don't know what the "ON DELETE" and "ON UPDATE" options mean, look them up on mysql.com

   v) Test it ;-> (maybe do "Task 5" 1st?!) Delete a row in a "parent" table and see if the deletion cascades as it should.

Task 4: Backup your precious database tables in *one* file

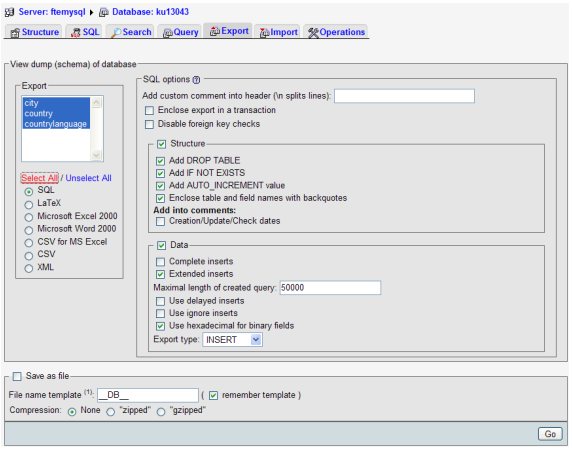
Thankfully now that the data exist in MySQL it's easy to reduce the database creation and population steps to one easy task by exporting the table data:

1) Open phpMyAdmin and locate the "export" tab from *your* database :

   i) Select *all* of your tables

   ii) Select "Add DROP TABLE" and "Add IF NOT EXISTS" so the backup is easy to restore.

   - "Extended inserts" shortens it slightly.



iii) Choose a file format (if not, you get a big SQL textarea that you can copy/paste to a file if necessary …)

iv) Click "Go" and save the file on your H: drive.

2) Examine the ***DBname*.sql** file (open the ZIP and unpack if necessary)

i) Open it in an editor ("gedit" or similar) & examine the structure, it's educational…

ii) Make sure it contains data and CREATE definitions for all the tables you exported.

- To restore everything you could now (provided the textarea is big enough!) copy/paste the file's contents into either SQL text area or upload the commands as-is.

<huge sigh of relief>

That's it for today ... if you have time you could continue to explore phpMyAdmin, the PHP manual or the MySQL documentation.