# University of London International Programmes

# CO2209 Database systems

# Coursework assignment 2 2016–2017

**General information**

## Important

**Your Coursework assignment should be submitted as a single PDF file, using the following file-naming conventions**:

FamilyName_SRN_COxxxxcw#.pdf (e.g. Zuckerberg_920000000_CO2209cw2.pdf)

- **FamilyName** is your family name (also known as last name or surname) as it appears in your student record (check your student portal)
- **SRN** is your Student Reference Number, for example 920000000
- **COXXXX** is the course number, for example CO2209, and
- **cw#** is either cw1 (coursework 1) or cw2 (coursework 2).

## Background

In Coursework assignment 1, we created a 'toy' database, consisting of four tables and a few dozen rows. Real databases are typically many orders of magnitude larger, in terms of numbers of relations, and the size of each relation. This impacts on a crucial aspect of database usage, 'performance' (basically, how fast queries are processed). In this Coursework assignment, the practical task will be to download such a database, and use it to learn more about real databases.

We're going to download the 'Mondial' database, which carries information about the countries of the world. This is supposedly based on the CIA's World Fact Book.

Go to the following website: [http://www.dbis.informatik.uni-goettingen.de/Mondial/].
Find the paragraph entitled 'Generating the Database under MySQL' and click on each of the following links:
http://www.dbis.informatik.uni-goettingen.de/Mondial/OtherDBMSs/mondial-schema-mysql.sql

http://www.dbis.informatik.uni-goettingen.de/Mondial/OtherDBMSs/mondial-inputs-mysql.sql

You don't download the data in the database directly, but rather download statements in the first file that will create the tables (called mondial-schema-mysql.sql), and then statements in the second file (called mondial-inputs-mysql.sql) that will populate them.

The second file is about 1.5 mbytes in size, and it consists of over 20,000 INSERT INTO statements. These files may be downloaded and submitted to a 'front end' processor for your database, if you are using one (that is, the statements themselves will be processed).

Alternatively, the whole files, which are displayed when you click their links, may be copied and pasted into a text editor (you will probably need to add a new file name extension to the files to do this, for example, changing mondial-inputs-mysql.sql to mondial-inputs-mysql.sql.txt). Then the statements in those files can be copied and pasted directly to the MySQL command line processor, if you are running MySQL in command line mode. (Of course, you will first do this with the schema creation file, and then with the inputs file.)

Note: This is a large database, which may take up to half an hour to download. If you are using Linux and encounter a problem with the download, visit the discussion board on the VLE and see if your problem has been answered there.

From this same website (http://www.dbis.informatik.uni-goettingen.de/Mondial/) download the documents that display the structure of the database: there are three, all of which carry the same information but display it differently:

A ‘Referential Dependency’ diagram:
-- [http://www.dbis.informatik.uni-goettingen.de/Mondial/mondial-abh.pdf]

An Entity/Relationship diagram:
-- [http://www.dbis.informatik.uni-goettingen.de/Mondial/mondial-ER.pdf]

A Relational Schema:
-- [http://www.dbis.informatik.uni-goettingen.de/Mondial/mondial-RS.pdf]

These documents let us see how the data in the separate tables is related; namely, which tables hold data relating to the same things. Note that the Relational Schema shown here is only a broad outline schema. It doesn't show datatypes for the attributes, or which attributes are Primary or Foreign Keys, or other constraints.

The Entity/Relationship diagram is a very simple one, which omits cardinality and participation constraints. Don't worry too much about these documents until you think you have understood the concepts of data dependency, functional dependency, keys and normalization. They will be useful for constructing SQL statements to answer queries that require you to know which tables are linked to which other tables.

Important note about the Mondial Database: the value of this database is that it is not a toy one. However, it is definitely out-of-date, and it was inaccurate even when first put up on the internet. (Remember: all large data sets must be assumed to be ‘dirty’.)

Additionally, its designers made at least one poor choice: they have field names (attributes or columns) which are the same as relation names. So, there is a relation called ‘Country’, and in some of the other relations, there is an attribute called ‘Country’. The ‘Country’ field of these other relations is a Foreign Key for Country.Code. It would have been a better idea to label these attributes ‘Country-Code’. We will look at a way to improve this.

Also, please note that the Mondial database (as implemented using MySQL) does NOT enforce Foreign Key integrity. Therefore, it would be possible to have a ‘Country’ field in a relation that has no matching ‘Code’ field in the relation Country.

## Coursework assignments

**NOTE:** If you have questions about any of these assignments, or if you need help in completing them, use the discussion board on the VLE.

### A1. Reporting your experience setting up this database or with databases In general.

This need not be longer than 200 words. (As a guideline, the first two paragraphs of this coursework assignment, ending 'CIA World Fact Book' is about 130 words.)

Write **EITHER** one of these reports – not both.

| |
|---|
| **A1(a)** If you had any problems downloading and setting up this database, write a short report, after you have set up and populated your database, describing this experience. |
| **A1(b**) If you had no difficulties downloading and installing this database, write a short report briefly describing your past experience with database management systems, including any courses you have taken relevant to this subject, and any practical experience (paid or otherwise) you have had with databases. If you have had no such experience, write a few sentences describing what you have heard about databases and their uses. |

**[3 hours (for downloading and creating, plus writing), 10 marks]**

**Tip:** in using MySQL, if your operating system is Windows, you may want to make the DOS Command Prompt window larger than the default value. Go here to see how to do this:

http://www.isunshare.com/windows-10/change-command-prompt-window-size-in-windows-10.html

(There are similar sites on the internet for earlier versions of Windows. You may have to reboot after doing this for the new window size to take effect.)

## A2. Compiling a description of the tables you have downloaded

This compilation will provide basic information about the database tables you have set up. If you get a job as a Database Administrator or you have to work with a database that you did not create yourself then you will have to do something like this first. Combined with the Referential Dependency diagram, E/R diagram and Relational Schemas you have already downloaded, you will have the materials you need to start understanding your database's structure.

You don't actually have to write anything here, but rather, just copy in the results of running some commands:

To do this, you will need the SQL commands
**SHOW TABLES**; and
**DESCRIBE** <tablename>; and
**SELECT COUNT(*) FROM** <tablename>.

Note that for '<tablename>' you will need to substitute the names of each of the tables listed by **SHOW TABLES.** Use of a word processor or a simple text editor can make this a very quick operation if you are working with MySQL directly from the DOS prompt. Note that for earlier versions of Windows, to paste into the Command Prompt, you may need to right-click – CTRL-V may not work.

It will be useful to be able to output what you see on the screen to an output file; you can use the command 'tee' to do this, as follows (user input in bold):

mysql> **TEE** D: OutputLog.txt – whatever shows on the screen is also copied to the file OutputLog.txt. For this example I have placed it on my D:disc, but it can be located anywhere you like.

mysql> **SHOW TABLES;** – information about the tables will be sent to OutputLog.txt as well as being shown on the screen;
mysql> **DESCRIBE BORDERS**;
mysql>**SELECT COUNT(*) FROM BORDERS;**
mysql> **SHOW INDEX FROM BORDERS;**
- and so on ... for the first *five* relations in this database (COUNTRY to DESERT);

mysql> **notee;** – turns it off;

**A2(a)** Include the requested information for the first five tables, plus the size of each of these tables, which you can find out by executing the commands below.

To see the size in megabytes of each of your tables, do this:

```
SELECT
    table_schema as `Database`,
    table_name AS `Table`,
    round(((data_length + index_length) / 1024 / 1024), 2) `Size in MB`
FROM information_schema.TABLES
WHERE table_schema = 'mondial'
ORDER BY (data_length + index_length) DESC;
```

**A2(b)** Answer the following questions:
(1) What is the total size of the Mondial database?
(2) What are the two largest relations in the Mondial database in terms of total bytes?
(3) For any two relations (in any database) is it the case that the relation with the largest cardinality must be the largest in terms of total bytes of data?
(4) For any two relations, is it the case that the relation with the largest degree (number of columns) must be the largest in terms of total bytes of data?
(5) If, given two relations, one is larger than the other in terms of both cardinality and degree, is it necessarily larger than the other, in terms of total bytes of data?

Please start this answer on a new page.

**[2 hours, 10 marks]**

## A3. Queries on the Mondial database

**A note about SQL:** SQL's tables do not conform completely to the definition of relations. In particular, the tables that result from a query can have duplicate tuples (rows), which in most cases is not what we want, and it violates the definition of 'relation'. To avoid this, always use the **DISTINCT** keyword, as in **SELECT DISTINCT …**

**Show not just your query but also the data set that result. Use 'LIMIT 10' at the end of your query if your query returns more than ten results.**

**A3(a)** What is the query that will list the names of all cities that have populations greater than the population of London? What happens if you leave out the **SELECT DISTINCT** requirement and type in only **SELECT**?

**A3(b)** What is the query that will list the names of all countries that have a GDP < 10,000 and an infant mortality greater than 5? **Show not just your query but also the data set that result.**

**A3(c)** What is the query that will list the names of all countries that are members of the World Trade Organization and have GDP's greater than 1,000,000? **Show not just your query but also the data set that result.**

**A3(d)** What is the query that will list the names of all countries that are NOT members of the World Trade Organization and have GDP's greater than 100,000? **Show not just your query but also the data set that result.**

**A3(e)** What is the query that will list the world's rivers, and for each one, the total number of countries where each is represented? (Hint: use **GROUP BY**). **Show not just your query but also the data set that result.**

**A3(f)** What is the query that will list the world's rivers that pass through at least 3 countries, and for each one, the total number of countries where it runs? (Hint: use **GROUP BY** and **HAVING**). **Show not just your query but also the data set that result.**

**[10 hours, 5 marks each. Total: 30 marks]**

## A4. Getting help/Finding out more

The rapid advance of computer technology, especially the internet, has profoundly affected the world of databases. Whereas someone graduating in 1987 did not see much substantial change in the database world for the next ten years, you will almost certainly see deep changes over the next decade. If your job involves a significant use of databases, you will need to keep up with developments in this field. In addition, you may well want to extend your knowledge of databases beyond the fundamentals provided in this course.

There are many online resources, which can help you keep up with developments in the field, and deepen your knowledge of the existing technology. This section of the Coursework assignment introduces you to some of them.

**A4 (a)** Search the internet for an online journal or forum, or other websites dealing with database issues, and find information on MySQL's GROUP_CONCAT feature. Describe in no more than 250 words what it does and list three web URLs where you found your information. Don't use the official MySQL manual.

Please start this answer on a new page.

**[2–4 hours, 10 marks]**

**A4 (b)** Go to YouTube and find video presentations on the subject of 'Big Data'. In no more than 250 words, write a short summary of the information they present. List the URL's of each video you watch and their running times. The total running times for your selections should **not** be less than 45 minutes; therefore, you can watch one video that runs for an hour, or four that run for 12 minutes each, just so long as the total adds up to at least 45 minutes.

Please start this answer on a new page.

**[2–4 hours, 10 marks]**

## B. Designing a Database

In this part of the Coursework assignment, we will represent a situation with an Entity/Relationship diagram, and then move from the diagram to a relational schema. (A relational schema is just the proposed relations for a database – their names, attributes, primary keys and foreign keys. It may also include data types and constraints, although ours will not.)

Being able to recognise which attributes or combinations of attributes make up a Candidate Key (something that uniquely identifies each tuple, i.e. is necessarily different for each tuple) is one of the skills that you need to learn in this course. It's not difficult, but it may take some practice. Some students get the wrong idea that the Primary Key is the 'most important' attribute or combination of attributes, 'what the table is giving information on'. This is not necessarily so.

A Candidate Key uniquely identifies a row (tuple). You can check your choice of Candidate Keys by asking, 'Could any two rows have the same values for their Candidate Keys, as I have chosen them?' If so, then, you have made the wrong choice.

A Candidate Key is a 'possible key'. If there can be more than one possible key for a table, then we have to choose one as the Primary Key. Note that a Primary Key is also a Candidate Key, although it's common to hear the phrase 'Candidate Key' used to refer only to those Candidate Keys that were not chosen to be the Primary Key. (A better name for unchosen Candidate Keys might be 'Alternate Keys').

Designating a Primary Key protects a table from one kind of integrity violation. Consider the database that holds your University of London student registration number. There will be one table where the attribute with this number is the Primary Key, with other attributes holding your name and other information about you. If the Primary Key integrity (also called 'Entity Integrity') was not enforced, then there could be two different people with the same registration number. (Another way of putting it is this: the Primary Key designation automatically enforces the UNIQUE constraint, but whereas you can designate more than one attribute (column) as UNIQUE, you can have only one Primary Key.

A School of Beauty Therapy offers various **Courses** leading to a Beauty Therapist Certificate.

Each **Course** – which has a unique name – has its own set of criteria for admission (a special entrance examination or a recognised diploma of secondary education, etc.). These are represented as a single text entry in the database, with its own fee, and its own method of allowing students to gain practical experience (some via work experience, others by a practical

session during the summer at the School's own facilities). A Course can exist in the database without any students being enrolled in it.

A typical **Course** might be called 'Natural Healing Therapy' that requires a diploma in secondary education, costs $4,500 and requires external work experience.

Every Course consists of a set of modules; some are compulsory and others optional, each carry a certain number of course credits. Each module has a name, a link to a syllabus and the modules, if any, which are prerequisite for it. A particular module is taught by one lecturer and, in a given year, a particular lecturer might teach several, one or no modules. (Note that a given module can be compulsory for one or more courses, but optional for others.)

The Course described above might consist of, among others, a Module in 'Aroma Therapy', (optional for this course), worth six credits, with a syllabus to be found at www.BeautyCollege/Syllabus/AromaTherapy.pdf, and have two prerequisite modules: Introduction to Therapy, and Safety with Chemicals. In the current year, Safety with Chemicals is taught by Irene Pulikoff.

We also want to record which lecturers can teach which modules – a lecturer may be recorded as being able to teach a module that he or she is not currentlyteaching that year. For instance, Irene Pulikoff can teach Aroma Therapy and Meditation as well as Safety with Chemicals.

A student in the database must be registered for at least one course. Optional modules, during a given year, may have no students registered for them. Courses may share modules – that is, two or more courses may require their students to take the same module.

Students are identified by a Student Number. We want to record the course they are enrolled on, the year they enrolled, the modules they have already attempted, the year they were examined in that module, and the grade they received. (Students may re-sit examinations in modules they fail.) A student can be registered for a course, but not (yet) registered for any optional modules.

For example, Kim Pak, whose Student Number is P3938, enrolled for the Aroma Therapy course in 2016, and passed 'Introduction to Therapy' and 'Safety with Chemicals' that same year, receiving 84% and 92% in those courses, respectively.

Lecturers are identified by Lecturer ID's. We want to record their names, date of birth, the modules they can teach, and the modules they are currently teaching. It is possible to have lecturers who are not teaching any modules currently, but all lecturers must be able to teach at least one module.

Modules are identified by a module name, and we want to record the number of credits a module is worth. A new module that has not yet started may not have any lecturers able to

teach it, but it will still be represented in the database. No module will be taught by more than one lecturer.

**B1.** Draw an Entity/Relationship diagram that expresses the relationships among the entity types described above. You may assume that the Entity Types are Course, Module, Student, and Lecturer. You need **not** indicate the attributes of each entity type, but you should show 'participation and cardinality constraints'. (In other words, show whether a relationship is necessary or optional for each participating entity type and if there are minimum or maximum numbers of participating entities in a relationship.) You may wish to review the *CO2209 Database systems* subject guide*, Volume 1*, pp.106–19, before starting this part of the Coursework assignment.

Please start this answer on a new page.

**[5 hours, 15 marks]**

**B2.** Design a fully normalized relational schema that can capture the data relationships expressed in your Entity-Relationship diagram. Be sure to indicate the primary and foreign keys of each relation. You need not indicate the data types of the attributes, nor other constraints on them. In order to provide an example of a relational schema, the first one has been done for you.

You may wish to review the *CO2209 Database systems* subject guide*, Volume 1*, pp.119–46, before starting this part of the Coursework assignment.

Please start this answer on a new page.

COURSE-MASTER-DATA

| CourseName | Criteria | Fee | WorkExperience |
|---|---|---|---|

Primary Key: CourseName

Foreign Keys: None

**[5 hours, 15 marks]**

**[Total 100 marks]**

## END OF COURSEWORK ASSIGNMENT 2

## APPENDIX I: E/R Modelling conventions

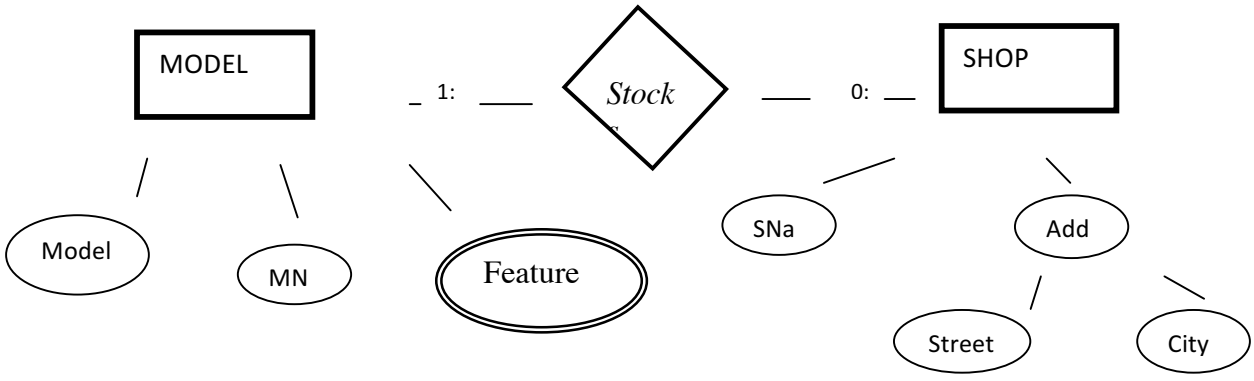
Use the following example as a model for your own E/R diagrams. (There are many variants of Entity/Relationship diagrams; this is one of the most common.) There are additional refinements to E/R diagrams with which you should be familiar; they are discussed in the *CO2209 Database systems* subject guide. In particular, make sure you know how entity sub-types and how 'weak entities' can be represented.

This illustrates part of a database dealing with shops that sell model airplanes.

For each model, there is a unique Model-Number, a name, and one or more 'features', such as 'not for children under 3', 'requires assembly', 'takes batteries', etc. For each shop there is a unique name and an address, which consists of a street, such as '34 Lyons Lane', and a City, such as 'Leeds'. We want to record these attributes, plus the *relationship* 'stocks' between shops and models.

A shop will not exist in the database unless it stocks at least one model, but a model may exist in the database even if no shop stocks it. In this example, these constraints are shown by 'minimum:maximum' numbers, where 'N' means there is no upper limit (the usual case). (Existence and cardinality constraints are sometimes indicated by 'crow's feet' and single vs double lines.)

Note that the 1:N or 0:N indicators are on the *opposite* side of the 'relationship' diamond to the entity types they are describing. This is sometimes called 'look-across' notation. If we wanted to show information about a relationship itself – for example, 'date-first-stocked' – we could either add an attribute-bubble to the relationship diamond, or treat 'stocked' as an entity type in itself (since we are going to make it into a table anyway). Both conventions are common.




Note: attribute names and entity-type names are normally given in the singular. Thus 'MODEL' not 'MODELS'.

## Appendix II: Generating a Relational Schema from an E/R diagram

To generate a relational schema from the diagram in **Appendix I**, we first create at least one relation for each entity type, whose Primary Key will be the entity-identifier attribute. (In this example, such attributes have an asterisk beside their names.) All attributes (including composite attributes, such as the 'Address' attribute for shops) that have only one value per entity instance, can go into this relation.

An attribute that can have multiple instances, such as the 'Feature' attribute for Models, must have a separate relation, whose attributes will be the entity-identifier and the appropriate attribute value, both of which will make up a composite Primary Key.

Finally, there must a separate relation for all relationships which are N:N, as this one is above. The key of this relation will be the entity-identifiers of the two participating entity types. (1:N relationships are rare but when they do occur, they can be implemented in the relation of the 'N' entity. For example, if a model could only be stocked in one shop, we could record that shop's name along with the model name in the model relation.)

So, we would have the following relations:

**MODEL**

**Attributes:** Model-Number, Mname
**Primary Key:** Model-Number

**MODEL-FEATURES**

**Attributes:** Model-Number, Feature
**Primary Key:** Model-Number + Feature

Note: an alternative – ***bad*** -- design here would be ,

**Attributes:** Model-Number, Feature-1, Feature-2, Feature-3, Feature-4…
**Primary-Key:** Model-Number

This is a very poor design for several reasons (although it's often one that beginners choose):

(1) We don't know how many features any given model will have. If a new model has more features than we have feature attributes, we will have to restructure the whole table by adding a new column. In the original design, this is never necessary. A model could have fifty features, and we would just add fifty rows to the Features table.

(2) Consider querying this table. In the original design, a typical SQL statement to find all models requiring assembly would be:

**SELECT** Model-Number **FROM** Model-Features **WHERE** Feature = 'requires assembly';

In the alternative design, we would have to write:

> **SELECT** Model-Number **FROM** MODEL-FEATURES **WHERE** Feature-1 = 'requires assembly' **OR** Feature-2 = 'requires assembly' **OR** Feature-3 = 'requires assembly' **OR** Feature-4 = 'requires assembly' ;
>
> An even worse design would be to create a single 'Feature' attribute, but holding all features as one giant string, like this:
>
> **Attributes:** Model-Number, Feature
>
> **Primary Key:** Model-Number

Be sure you understand the differences between these three designs.

Not only would this make a very ugly print-out, but it would require us to use an inefficient SQL feature (LIKE %substring%) to form the query above. In addition, it won't work at all if the attribute in question is numeric or of type date.

**SHOP**

> **Attributes:** Sname, Street, City, Post-Code
> **Primary Key:** Sname

Note: 'Address' is 'flattened' here. The relational model has no provision for recognising composite attributes. That is, you couldn't have a relation header that looked like this, even though you might design a printed or on-screen form this way:

| | Address | | |
|---|---|---|---|
| Sname | Street | City | Post-Code |

**MODEL-IN-SHOP**

> **Attributes:** Model-Number, Sname
>
> **Primary Key**: Model-Number + Sname