COMP1314 Coursework

Oran Keating (ok1g24@soton.ac.uk)

December 2024

1 Structured Data

1.1 Ex1

The bash script below converts both students.xml and faculty.xml into CSV formats. Its first two "if" statements are to ensure the correct number of command line arguments are given and that, when give, they are the correct arguments. The final two "if" statements actually convert the files into CSV's. The code used to convert the scripts is by no means the most efficient way of conversion nor is it able to be used for any xml script hence the two different versions for facutly.xml and student.xml. I will explain what each part of the command to actually convert the files does below the script.

```bash
#!/bin/bash

#Ensuring correct No. of command line arguments are given
if [[ $# != 1 ]]; then
  echo "Error: one command Line argument must be provided"
  exit 1
fi


if [[ "$1" != "faculty.xml" && "$1" != "students.xml" ]]; then
  echo "Error: command line arguemt must be either faculty.xml or students.xml"
  exit 1
fi

#Processing data and outputting to CSV
if [[ "$1" == "faculty.xml" ]]; then
  echo "Writing to faculty.csv..."
  echo "faculty,building,room,capacity" > faculty.csv
  grep -v xml "$1" | sed -E 's/<\/?[a-z]*>//g' | awk '{$1=$1; print}' | tr -d '\r' | sed '/./ N; s/\n/,/g' | sed '/./ N; s/\n/,/g' | awk 'NF {printf"%s\n",$0}' >> faculty.csv
fi

if [[ "$1" == "students.xml" ]]; then
  echo "Writing to students.csv..."
  echo "student_name,student_id,student_email,programme,year,address,contact,module_id,module_name,module_leader,lecturer1,lecturer2,faculty,building,room,exam_mark,coursework1,coursework2,coursework3" > students.csv
  grep -v xml "$1" | sed 's|<[^>]*\/>|<1>NULL<\/1>|g' | awk '{$1=$1;print}'| sed -e 's|\(^[^< ]\)| \1|g' | tr -d '\r\n' | sed -e 's|\([^>]*,[^<]*\)|"\1"|g' | sed 's|<\/student>|\n|g' | sed 's|<[^\/]*>||g' | sed 's|<\/[^>]*>|,|g' | sed 's|,$||g' >> students.csv

fi

echo "Done"
```

Listing 1: XML to CSV script

Code	Explanation
grep -v xml "$1"	Removing XML version tag
sed -E ’s/<\/?[a-z]*>//g’	Removing <> and </> tags
awk ’{$1=$1; print}’	Removing leading and trailing whitespace
tr -d ’\r ’	Removes \r as file has been opened in both Linux and windows
sed ’/./ N; s/\n /,/g’	Combines 2 lines into one by replacing newlines with commas. Only does this for the non-empty lines as specified by the /./ and takes into account the next line too due to the "N;"
sed ’/./ N; s/\n /,/g’	Same as above as we need to combine a total of 4 lines into one
awk ’NF \{printf"\%s\\n",\$0\}’	Removes 2 blank lines in between each line of text by printing each line verbatim with a \n at the end to separate them
> faculty.csv	Piping output into file called faculty.csv

Table 1: Explanation of code for faculty.xml

Note: sed commands are not fully shown due to some characters being delimiters in LaTeX verbatim environments. However, they are in the correct order.	
Code	Explanation
grep -v xml "$1"	Removing XML version tag
sed ...	Replacing empty xml tags with <1>NULL<> to be able to identify them later
awk ’{ $1 = $1 ; print } ’	Removing sorrounding whitespace
sed ...	Adds a space to any line that does not start with "<". This means that when later formatting the address line there remains a space when two lines are merged into one.
tr -d ’\ r \ n ’	Removes carriage returns and newlines
sed -e ...	Puts "" around text inside tags that contains a comma. This is so the comma can remain in the CSV file without it being parsed as a delimiter.
sed ...student...	Replaces closing student tag with a newline to separate each student. Allows each student to have their own row when it eventually becomes a CSV file.
sed ...	Removes all opening XML tags.
sed ...	Replaces all closing XML tags with a comma.
sed ’s|,$||g’	Removes all commas at the end of a line to properly seperate each student
>> students.csv	Specifically appending to students.csv as the header line is written previously

Table 2: Explanation of code for students.xml

1.2 Ex2

```
#!/bin/bash

if [[ "$#" -ne 2 ]]; then
        echo "Error: Two command line arguments must be provided"
        exit 1
fi


if [[ "$1" != "students.csv" ]]; then
        echo "Error: First command line argument must be students.csv"
        exit 1
fi

if [[ "$#" == 2 && "$2" != *.txt ]]; then
  echo "Error: The second argument must be a text file"
  exit 1
fi

file_name="$2"

echo "Writing to $file_name..."

cut -d, -f1,2 $1 | tail -n +2 | sort -t, -k1,1 | uniq -f1 | cut -d, -f1 > $file_name

echo "Done"
```

Listing 2: CSV to TXT script

Code	Explanation
cut -d, -f1,2 $1	Splits the file given by the first command line argument by commas and takes the first and second row
tail -n +2	Removes the first line from the file as the file contains the row headers
sort -t, -k1,1	Sorts by first column using a comma as the field separator
uniq -f1	deletes all duplicate student_ids as it skips the first field
cut -d, -f1	Splits by comma and takes the first row
> $file_name	writes output to file

Table 3: Explanation of code for generating a list of names from a csv

2 Relational Model

2.1 Ex3

Students relation:
Students(student_name, student_id, student_email, programme, year, address, contact_number, module_id, module_name, module_leader, lecturer1, lecturer2, faculty, building, room, exam_mark, coursework1, coursework2, coursework3)

Faculty relation:
Faculty(faculty,building,room,capacity)

2.2 Ex4

Assumptions

The following are assumptions that have been made about the wider domain of the dataset:

- Students can not take more than one program
- Students do not have more than one contact number
- The uni will never re-use email addresses
- More than one faculty can use the same room in the same building
- A module will only ever use the same room.
- A module will only ever have the same lead and lecturers
- A program is not run by more than one faculty
- Only one faculty can run a module but multiple programs can take that module
- Lecturer 1 and Lecturer 2 are not two distinct attributes and their order can be swapped
- Coursework1, 2 and 3 are three distinct attributes who's order matters and can not be swapped

Minimal set of functional dependencies

```
Students Relation:
student_id -> student_name
student_id -> student_email
student_id -> programme
student_id -> year
student_id -> address
student_id <-> contact
student_email <-> contact

student_id, module_id -> exam_mark
student_id, module_id -> coursework1
student_id, module_id -> coursework2
student_id, module_id -> coursework3

module_id -> module_name
module_id -> module_leader
module_id -> lecturer1
module_id -> lecturer2
module_id -> faculty
module_id -> building
module_id -> room

programme -> faculty

Faculty Relation:
building, room -> capacity
```

2.3 Ex5

Faculty relation candidate keys:

- Composite Primary Key: (faculty,building,room)

Students relation candidate keys:

- Composite Primary Key: (student_id,module_id)
- Composite Primary Key: (student_email,module_id)
- Composite Primary Key: (contact_number,module_id)

2.4 Ex6

Faculty relation:
Primary composite key: (faculty,building,room)
This is because this is the only candidate key

Students relation:
Primary composite key: (student_id, module_id)
This is because an id number is a common and known way to uniquely identify people in a large organization and deviating from the norm is likely to cause confusion. In addition students may change their email or contact number over time and this will result in have to update the database however they will never change their id.

3 Normalisation

3.1 Ex7

The data is not in first normal from. There are repeating groups such as (lecturer1, lecturer2) and (coursework1,coursework2,coursework3)

The set of relations is as follows:

- Students(student_name, student_id, student_email, programme, year, address, contact_number, module_id, module_name, module_leader, faculty, building, room, exam_mark)
- Lecturers(module_id, lecturer)
- Courseworks(student_id, module_id, coursework_number, coursework_mark)
- Faculty(faculty,building,room,capacity)

3.2 Ex8

In the existing dataset there is missing data and this has been replaced with NULL data. As the lecturer number was not of importance it was able to be decomposed into a table with just module_id and lecturer however because coursework1, 2, 3 detonated which coursework the mark was for a coursework_number column is required.

3.3 Ex9

Partial dependency's in faculty relation:

- building, room -> capacity

New relations to be made as a result:

- Room_Capacity(building,room,capacity)
- Faculty_Room(faculty,building,room)

Partial dependency's in students relation:

- module_id -> module_name
- module_id -> module_leader
- module_id -> faculty
- module_id -> building
- module_id -> room
- student_id -> student_name
- student_id -> student_email
- student_id -> programme
- student_id -> year
- student_id -> address
- student_id -> contact

New relations to be made as a result:

- student_info(student_name, student_id, student_email, programme, year, address, contact)
- students_modules(student_id, module_id, exam_mark)
- Modules(module_id, module_name, module_leader, faculty, building, room)

3.4 Ex10

Decomposing faculty: In the faculty relation there is one partial dependency. Capacity is only dependent on building and room but not faculty. To resolve this the following relations can be created and the existing faculty relation removed

- Room_Capacity(building,room,capacity)
- Faculty_Room(faculty,building,room)

Decomposing students: Students can first be broken up into the following two relations:

- Modules(module_id, module_name, module_leader, faculty, building, room)
- Students(student_name, student_id, student_email, programme, year, address, contact, module_id, exam_mark

However in the second of these two relations there still exists partial dependencies. As the key is a composite key consisting of (student_id, module_id) all attributes other than exam mark rely on solely student_id. We therefore have to further decompose this relation into two new ones.

- student_info(student_name, student_id, student_email, programme, year, address, contact)
- students_modules(student_id, module_id, exam_mark)

List of current relations, fields and types

<table>
<tr><th colspan="2">room_capacity relation</th></tr>
<tr><td colspan="2">Primary Key: building, room</td></tr>
<tr><td>Field</td><td>Type</td></tr>
<tr><td>building</td><td>str</td></tr>
<tr><td>room</td><td>str</td></tr>
<tr><td>capacity</td><td>int</td></tr>
</table>

<table>
<tr><th colspan="2">faculties_room relation</th></tr>
<tr><td colspan="2">Primary Key: building, room</td></tr>
<tr><td colspan="2">Foreign Keys: (building, room) -> room_capacity</td></tr>
<tr><td>Field</td><td>Type</td></tr>
<tr><td>faculty</td><td>str</td></tr>
<tr><td>building</td><td>str</td></tr>
<tr><td>room</td><td>str</td></tr>
</table>

<table>
<tr><th colspan="2">student_info relation</th></tr>
<tr><td colspan="2">Primary Key: student_id</td></tr>
<tr><td>Field</td><td>Type</td></tr>
<tr><td>student_name</td><td>str</td></tr>
<tr><td>student_id</td><td>str</td></tr>
<tr><td>student_email</td><td>str</td></tr>
<tr><td>programme</td><td>str</td></tr>
<tr><td>year</td><td>int</td></tr>
<tr><td>address</td><td>str</td></tr>
<tr><td>contact</td><td>str</td></tr>
</table>

<table>
<tr><th colspan="2">modules relation</th></tr>
<tr><td colspan="2">Primary Key: module_id</td></tr>
<tr><td colspan="2">Foreign Keys: (building, room) -> room_capacity</td></tr>
<tr><td>Field</td><td>Type</td></tr>
<tr><td>module_id</td><td>str</td></tr>
<tr><td>module_name</td><td>str</td></tr>
<tr><td>module_leader</td><td>str</td></tr>
<tr><td>faculty</td><td>str</td></tr>
<tr><td>building</td><td>str</td></tr>
<tr><td>room</td><td>str</td></tr>
</table>

student_modules relation	
Primary Key: student_id, module_id	
Foreign Keys: student_id -> student_info, module_id -> modules	
Field	Type
student_id	str
module_id	str
exam_mark	int

lecturers relation	
Primary Key: module_id, lecturer	
Foreign Keys: module_id -> modules	
Field	Type
module_id	str
lecturer	str

courseworks relation	
Primary Key: student_id, module_id, coursework_number	
Foreign Keys: student_id -> student_info, module_id -> modules	
Field	Type
student_id	str
module_id	str
coursework_number	int
coursework_mark	int

3.5 Ex11

For the faculty relation the primary key was a composite key consisting of building,room,faculty however a partial dependency existed as capacity only relied on building,room and not faculty. Decomposing it into two tables both with the primary keys of building,room ensures the table is in 2NF.

For the student relation two separate decompositions had to be made before the relations were in 2NF. One to split students and modules and another to split the new students relation into student_info and student_modules.

When decomposing in both case I removed the minimum amount of attributes possible to a new table in order to keep the tables as simple as possible and not risk violating 2NF.

3.6 Ex12

There is one translative dependency in the modules relation

- module_id -> building, room -> faculty

3.7 Ex13

To fix the transitive dependency the "faculty" atribute is removed from the "modules" relation. No new relations need to be created as there already exists a relation linking "faculty" with "building" and "room", that being the "faculties_rooms" relation. Below is the adjusted relation.

modules Relation	
Primary Key: student_id, module_id	
Field	Type
module_id	str
module_name	str
module_leader	str
building	str
room	str

4 Modelling

4.1 Ex14

student_info Relation	
Primary Key: student_id	
Field	SQLite Datatype
student_name	TEXT
student_id	INTEGER
student_email	TEXT
programme	TEXT
year	INTEGER
address	TEXT
contact	TEXT

student_modules Relation	
Primary Key: student_id, module_id	
Field	SQLite Datatype
student_id	INTEGER
module_id	TEXT
exam_mark	INTEGER

modules Relation	
Primary Key: module_id	
Field	SQLite Datatype
module_id	TEXT
module_name	TEXT
module_leader	TEXT
building	TEXT
room	TEXT

lecturers Relation	
Primary Key: module_id, lecturer	
Field	SQLite Datatype
module_id	TEXT
lecturer	TEXT

courseworks Relation	
Primary Key: student_id, module_id, coursework_number	
Field	SQLite Datatype
student_id	INTEGER
module_id	TEXT
coursework_number	INTEGER
coursework_mark	INTEGER

room_capacity Relation	
Primary Key: building, room	
Field	SQLite Datatype
building	TEXT
room	TEXT
capacity	TEXT

faculties_room Relation	
Primary Key: building, room	
Field	SQLite Datatype
faculty	TEXT
building	TEXT
room	TEXT

4.2 Ex15

```
.import students.csv studentscsv
.import faculty.csv facultycsv
.output ex15.sql
.dump
```

Listing 3: SQL Code to import students.csv and faculty.csv

4.3 Ex16

The NULLIF SQL command will convert any "NULL" strings it finds to actual null values in the tables.

```
PRAGMA foreign_keys = ON;

--FACULTY RELATIONS

CREATE TABLE IF NOT EXISTS room_capacity (
    building TEXT NOT NULL,
    room TEXT NOT NULL,
    capacity INTEGER DEFAULT 0 CHECK (capacity >= 0),
    PRIMARY KEY (building, room)
    );

CREATE TABLE IF NOT EXISTS faculty_room (
    building TEXT NOT NULL,
    room TEXT NOT NULL,
    faculty TEXT NOT NULL,
    PRIMARY KEY (building, room)
    );


INSERT INTO room_capacity (building, room, capacity)
SELECT DISTINCT building, room, capacity
FROM facultycsv;

INSERT INTO faculty_room (building, room, faculty)
SELECT DISTINCT building, room, faculty
FROM facultycsv;


--STUDENT RELATIONS

CREATE TABLE IF NOT EXISTS student_info (
student_name TEXT,
student_id INTEGER NOT NULL CHECK (student_id >= 0),
student_email TEXT,
programme TEXT,
year INTEGER CHECK (year > 0),
address TEXT,
contact TEXT,
PRIMARY KEY (student_id)
);

INSERT INTO student_info (student_name, student_id, student_email, programme, year, address, contact)
SELECT DISTINCT NULLIF(student_name, 'NULL'),
                student_id,
                NULLIF(student_email, 'NULL'),
                NULLIF(programme, 'NULL'),
                NULLIF(year, 'NULL'),
                NULLIF(address, 'NULL'),
                NULLIF(contact, 'NULL')
FROM studentscsv;

CREATE TABLE IF NOT EXISTS modules (
module_id TEXT NOT NULL,
module_name TEXT NOT NULL,
module_leader TEXT,
building TEXT,
room TEXT,
PRIMARY KEY (module_id),
FOREIGN KEY (building, room) REFERENCES room_capacity(building, room)
);

INSERT INTO modules (module_id, module_name, module_leader, building, room)
SELECT DISTINCT module_id, module_name, NULLIF(module_leader, 'NULL'), NULLIF(building, 'NULL'), NULLIF(room, 'NULL')
FROM studentscsv;

CREATE TABLE IF NOT EXISTS student_modules (
student_id INTEGER NOT NULL,
module_id TEXT NOT NULL,
exam_mark INTEGER CHECK (exam_mark >= 0),
PRIMARY KEY (student_id, module_id),
FOREIGN KEY (student_id) REFERENCES student_info(student_id),
FOREIGN KEY (module_id) REFERENCES modules(module_id)
);

--ROWS should be inserted regardless of NULL exam_mark as student still takes the module
INSERT INTO student_modules (student_id, module_id, exam_mark)
SELECT DISTINCT student_id, module_id, NULLIF(exam_mark, 'NULL')
FROM studentscsv;
```

```sql


CREATE TABLE IF NOT EXISTS lecturers (
module_id TEXT NOT NULL,
lecturer TEXT,
PRIMARY KEY (module_id, lecturer)
FOREIGN KEY (module_id) REFERENCES modules(module_id)
);

INSERT INTO lecturers (module_id, lecturer)
SELECT DISTINCT module_id, lecturer1 AS lecturer
FROM studentscsv
WHERE lecturer1 != 'NULL';

INSERT INTO lecturers (module_id, lecturer)
SELECT DISTINCT module_id, lecturer2 AS lecturer
FROM studentscsv
WHERE lecturer2 != 'NULL';

CREATE TABLE IF NOT EXISTS courseworks (
student_id INTEGER NOT NULL,
module_id TEXT NOT NULL,
coursework_number INTEGER CHECK (coursework_number > 0),
coursework_mark INTEGER CHECK (coursework_mark >= 0),
PRIMARY KEY (student_id, module_id, coursework_number),
FOREIGN KEY (student_id) REFERENCES student_info(student_id),
FOREIGN KEY (module_id) REFERENCES modules(module_id)
);

INSERT INTO courseworks (student_id, module_id, coursework_number, coursework_mark)
SELECT DISTINCT student_id, module_id, 1 AS coursework_number, NULLIF(coursework1, 'NULL') AS coursework_mark
FROM studentscsv;

INSERT INTO courseworks (student_id, module_id, coursework_number, coursework_mark)
SELECT DISTINCT student_id, module_id, 2 AS coursework_number, NULLIF(coursework2, 'NULL') AS coursework_mark
FROM studentscsv;

INSERT INTO courseworks (student_id, module_id, coursework_number, coursework_mark)
SELECT DISTINCT student_id, module_id, 3 AS coursework_number, NULLIF(coursework3, 'NULL') AS coursework_mark
FROM studentscsv;
```

Listing 4: SQL Code to create and populate the tables in the database

5 Querying

5.1 Ex17

```sql
SELECT building, SUM(capacity)
FROM room_capacity
GROUP BY building;
```

5.2 Ex18

```sql
SELECT student_info.student_id, student_info.student_name, AVG(student_modules.exam_mark) AS avg_exam_mark
FROM student_info
JOIN student_modules ON student_info.student_id = student_modules.student_id
WHERE exam_mark IS NOT NULL AND student_info.year = 1 AND student_info.programme = 'Computer Science'
GROUP BY student_info.student_id
ORDER BY avg_exam_mark DESC;
```

5.3 Ex19

```sql
SELECT module_id, module_leader, faculty, MAX(avg_marks)
FROM(
    SELECT m.module_id, m.module_leader, fr.faculty,
        AVG(
            CASE
                WHEN sm.exam_mark IS NOT NULL OR cw.coursework_mark IS NOT NULL
                THEN sm.exam_mark + cw.coursework_mark
                WHEN sm.exam_mark IS NOT NULL OR cw.coursework_mark IS NULL
                THEN sm.exam_mark
                WHEN sm.exam_mark IS  NULL OR cw.coursework_mark IS NOT NULL
                THEN cw.coursework_mark
            END
        ) AS avg_marks
    FROM modules m
    JOIN faculty_room fr ON (m.building = fr.building AND m.room = fr.room)
    JOIN student_modules sm ON m.module_id = sm.module_id
    JOIN courseworks cw ON (m.module_id = cw.module_id AND sm.student_id = cw.student_id)
    GROUP BY m.module_id, m.module_leader, fr.faculty
)
GROUP BY faculty;
```

5.4 Ex20

```sql
SELECT m.module_id, rc.building, rc.room
FROM modules m
JOIN room_capacity rc on (m.building = rc.building AND m.room = rc.room)
JOIN student_modules sm ON m.module_id = sm.module_id
GROUP BY m.module_id, rc.building, rc.room
HAVING COUNT(sm.student_id) > rc.capacity;
```