

# COMP1314 Coursework

James Worsley (jw5g25@soton.ac.uk)

November - December 2025

## 1 Relational Model

### ex4

#### Faculties

FACULTIES(faculty,building,room,capacity,lecturer\_email,lecturer\_firstname,lecturer\_surname)

#### Students

STUDENTS(student firstname,student surname,student id,student email,year,address,contact number,module id,module name,leader,  
lecturer1,lecturer2,exam mark,coursework1,coursework2,coursework3)

### ex5

#### Faculties

*lecturer\_email* → *lecturer\_firstname*  
*lecturer\_email* → *lecturer\_surname*  
*lecturer\_email* → *faculty*

*building, room* → *capacity*

#### Students

*studentid* → *studentfirstname*  
*studentid* → *studentsurname*  
*studentid* → *studentemail*  
*studentid* → *year*  
*studentid* → *contactnumber*  
*studentid* → *moduleid*

*moduleid* → *modulename*  
*moduleid* → *leader*  
*moduleid* → *lecturer1*  
*moduleid* → *lecturer2*  
*moduleid* → *coursework1*  
*moduleid* → *coursework2*  
*moduleid* → *coursework3*

*studentid, module* → *exammark*

### ex6

#### Faculties candidate keys

lecturer\_email, building and room is the primary key because it is used for only one lecturer within the faculty, although they may teach in multiple rooms.

#### Students candidate keys

student\_id and module\_id is the primary key because it corresponds to only one student, although they may be enrolled on multiple modules.

## 2 Normalisation

### ex7

#### Required for first normal form

- Elements are atomic
- No repeating groups

#### Making the data first normal form

The faculties relation is already in first normal form because every relation is atomic (each element is only one piece of data) and there are no repeating groups (each column is unique). We can keep our relation:

FACULTIES(faculty,building,room,capacity,lecturer\_email,lecturer\_firstname,lecturer\_surname)

However, the students relation is not first normal form. Firstly, the address column is not atomic because it contains the postcode and street name. Secondly, there are repeated groups. The lecturer columns are repeated (lecturer1, lecturer2) and the coursework columns are repeated (coursework1, coursework2, coursework3). Therefore, we need a new set of relations:

STUDENTS(student\_firstname,student\_surname,student\_id,student\_email,year,street,postcode,contact\_number)

MODULES(module\_id, module\_name, leader)

STUDENTMODULES(student\_id, module\_id, exam\_mark)

LECTURERMODULES(lecturer\_email, module\_id)

COURSEWORKMARKS(coursework\_id, student\_id, module\_id, mark)

### ex8

#### Partial-key dependencies

A partial key is when one of the non-key columns depends on only a part of a composite key. There are partial-key dependencies in our data:

- *building, room → capacity*
- *lecturer\_email → lecturer\_firstname, lecturer\_surname*
- *building → faculty*
- *lecturer\_email → faculty*

#### Required for second normal form

- No partial key dependencies (as described above)

#### Our data in second normal form

FACULTIES(faculty)

BUILDINGS(building, faculty)

ROOMS(building, room, capacity)

LECTURERS(lecturer\_email,lecturer\_firstname,lecturer\_surname)

LECTURERFACULTY(lecturer\_email, faculty)

STUDENTS(student\_firstname,student\_surname,student\_id,student\_email,year,street,postcode,contact\_number)

MODULES(module\_id, module\_name, leader)

STUDENTMODULES(student\_id, module\_id, exam\_mark)

LECTURERMODULES(lecturer\_email, module\_id)

COURSEWORKMARKS(coursework\_id, student\_id, module\_id, mark)

## ex9

### Transitive dependencies

A transitive dependency is when a non-key attribute depends on another non-key attribute, which in turn depends on the primary key.

### Required for third normal form

- For every non-trivial functional dependency,  $A \rightarrow B$ , A is a superkey or B is a prime attribute
- Attributes are determined only by the keys
- No transitive dependencies

### Our data in third normal form

As it stands, our data is in third normal form. **Before** we normalised to second normal form, we had the following transitive dependency:

$(building, room) \rightarrow lecturer\_email \rightarrow faculty$

However, this transitive dependency was dealt with when we converted our data to second normal form. Therefore, our relations remain the same:

FACULTIES(faculty)  
BUILDINGS(building, faculty)  
ROOMS(building, room, capacity)  
LECTURERS(lecturer\_email, lecturer\_firstname, lecturer\_surname)  
LECTURERFACULTY(lecturer\_email, faculty)  
STUDENTS(student\_firstname, student\_surname, student\_id, student\_email, year, street, postcode, contact\_number)  
MODULES(module\_id, module\_name, leader)  
STUDENTMODULES(student\_id, module\_id, exam\_mark)  
LECTURERMODULES(lecturer\_email, module\_id)  
COURSEWORKMARKS(coursework\_id, student\_id, module\_id, mark)

### 3 Modelling

ex11

#### FACULTIES

Attribute	SQLite Data Type
faculty	TEXT

#### BUILDINGS

Attribute	SQLite Data Type
building	TEXT
faculty	TEXT

#### ROOMS

Attribute	SQLite Data Type
building	TEXT
room	TEXT
capacity	INTEGER

#### LECTURERS

Attribute	SQLite Data Type
lecturer_email	TEXT
lecturer_firstname	TEXT
lecturer_surname	TEXT

#### LECTURERFACULTY

Attribute	SQLite Data Type
lecturer_email	TEXT
faculty	TEXT

#### STUDENTS

Attribute	SQLite Data Type
student_firstname	TEXT
student_surname	TEXT
studentID	INTEGER
student_email	TEXT
year	INTEGER
street	TEXT
postcode	TEXT
contact_number	TEXT

## MODULES

Attribute	SQLite Data Type
moduleID	TEXT
module_name	TEXT
leader	TEXT

## STUDENTMODULES

Attribute	SQLite Data Type
studentID	INTEGER
moduleID	TEXT
exam_mark	INTEGER

## LECTURERMODULES

Attribute	SQLite Data Type
lecturer_email	TEXT
moduleID	TEXT

## COURSEWORKMARKS

Attribute	SQLite Data Type
studentID	INTEGER
moduleID	TEXT
courseworkID	INTEGER
mark	INTEGER

## 4 Querying

ex13

```
SELECT
    STUDENTS.studentID,
    STUDENTS.student_firstname,
    STUDENTS.student_surname,
    AVG(STUDENTMODULES.exam_mark) AS average
FROM STUDENTS
JOIN STUDENTMODULES ON STUDENTS.studentID = STUDENTMODULES.studentID
WHERE STUDENTS.year = 1 AND STUDENTMODULES.moduleID LIKE '%COMP%'
GROUP BY STUDENTS.studentID, STUDENTS.student_firstname, STUDENTS.student_surname
ORDER BY average DESC;
```

ex14

```
WITH ALLMARKS AS (
    SELECT moduleID, exam_mark AS mark
    FROM STUDENTMODULES
    WHERE exam_mark IS NOT NULL
)
UNION ALL
SELECT moduleID, mark AS mark
FROM COURSEWORKMARKS
WHERE mark IS NOT NULL
),
MODULEAVGS AS (
    SELECT MODULES.moduleID, MODULES.leader, LECTURERFACULTY.faculty, AVG(ALLMARKS.mark) AS moduleAVG
    FROM MODULES
    JOIN LECTURERFACULTY ON MODULES.leader = LECTURERFACULTY.lecturer_email
    JOIN ALLMARKS ON MODULES.moduleID = ALLMARKS.moduleID
    GROUP BY MODULES.moduleID, MODULES.leader, LECTURERFACULTY.faculty
)
SELECT *
FROM MODULEAVGS ma
WHERE moduleAVG = (
    SELECT MAX(moduleAVG)
    FROM MODULEAVGS
    WHERE faculty = ma.faculty
);
```

ex15

```
WITH MODULESIZE AS(
    SELECT moduleID, COUNT(studentID) as numStudents
    FROM STUDENTMODULES
    GROUP BY moduleID
)
SELECT MODULESIZE.moduleID, BUILDINGS.building, ROOMS.room
FROM MODULESIZE
JOIN LECTURERMODULES ON MODULESIZE.moduleID = LECTURERMODULES.moduleID
JOIN LECTURERFACULTY ON LECTURERMODULES.lecturer_email = LECTURERFACULTY.lecturer_email
JOIN BUILDINGS ON LECTURERFACULTY.faculty = BUILDINGS.faculty
JOIN ROOMS ON BUILDINGS.building = ROOMS.building
WHERE MODULESIZE.numStudents > ROOMS.capacity
ORDER BY MODULESIZE.moduleID, BUILDINGS.building, ROOMS.room;
```