Design Document for RGU Course Database

I. Introduction

The use of Neo4j to build a database for the university course information presents a number of advantages, along with challenges that would not be present in a standard relational database. Prior to designing the schema, the following requirements were stated:

1) Scalability. It must be possible to add new courses, staff and student cohorts to the existing schema without altering the overall graph structure. In practice, this would be achieved by adding additional nodes.
2) Addition of new data. Along with adding new properties, it should be possible to add new data to the existing schema. The given use case was to add data for new academic years.
3) Flexible querying. Two example queries were specified.

The briefing for the design was quite high level so a conservative approach was taken for design, as such the following principles and assumptions were identified:

A Minimise data redundancy. Unlike a relational database, it is not necessarily practical/possible to normalise the data, but redundancy should be as low as possible without disproportionately affecting the other criteria.
B Maximise readability. As the final user of the database was not specified, it is assumed it will be accessed by individuals with varying levels of experience (a well-designed database is generally more readable anyway).
C It was assumed that the requirement to add data for future academic years also required all data for previous years to be retained.
D It must be possible to change any foreseeable attribute. For example, if a modules semester switches for a given course (but not for another if they share a common course), it must be possible to update this and keep a record of the previous value. It should be noted that there is probably no correct design as anything presented is a necessary compromise between requirements.

II. Design Choice

The above considered, two overall designs were devised.

A. Design 1 (discounted)

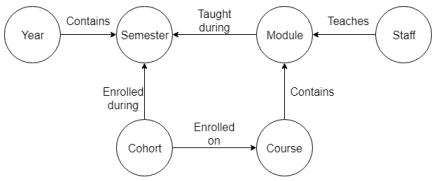

Fig. 1. Low relationship but high node-count design with normalised time properties

This design echoes the principles of relational database design and attempts to normalise the date-related properties with a timeline tree [1] so ensures date entries do not need to be repeatedly entered when new data is added (assuming none of the relations contain date attributes). This however, comes at the expense of node redundancy. Each node would require duplication in this model as new data is added. For example if module M1 was taught during semester 1 for course C1, and semester 2 for course C2, the database would contain this subgraph:

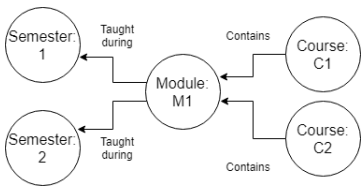

Fig. 2. subgraph from design 1 showing the need for either duplicate nodes or date-related relationship attributes

In this case, it is impossible to identify which course receives teaching for the module on a given semester without adding date attributes to the relationships. This completely negates the need for a timeline tree. Furthermore the requirement to duplicate nodes goes against the principle of minimising redundency. Adding a new year to this database would likely require a duplicate version of many nodes (essentially re-loading the database for every new year). Clearly unless normalisation of date-related attibutes is absolutely critical, which there is no reason to believe it is, a different design is necessary.

B. Design 2 (Adopted for the final design)

The full data model is as follows:

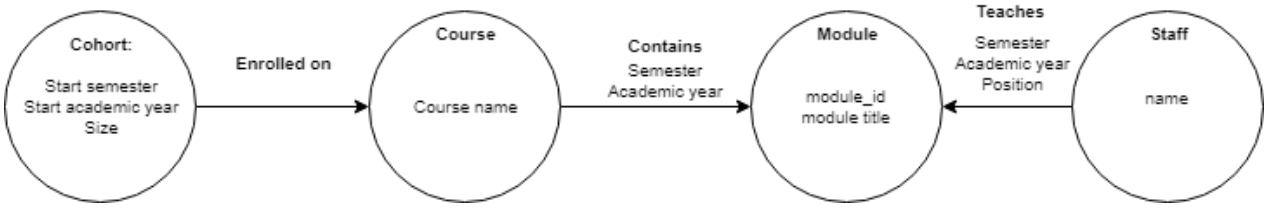

Fig. 3. Design of the proposed model

Unlike the previous option which would have a high node-count, this graph has a high relationship count. Although this leads to duplicated time data, it allows for any nodes to be re-mapped to any other eligible node as teaching schedules/course agendas change over time by adding new relationships.This design meets the requirements/principles outlined in the introduction as follows:

Mandatory criteria

1) It is relatively simple to add a new course, module or staff member. They are simply added as a new node and only need to be added once (ie they do not require separate entries for every academic year). The associated relationships can then be entered to link nodes up.
2) Adding data for a new academic year is achieved by simply adding new relationships between the existing nodes. This maintains a simpler graph with fewer nodes at the cost of having to re-enter the year and semester every time a new relationship is created. This was assessed to be a fair compromise. For example, every year, each staff member must be re-assigned to their respective courses creating a new relationship between the staff member and their respective courses (even if they continue to teach the same courses as the previous year). For example, if Ines Arana continues to teach CMM510 in the 2018 academic year, we need to add a relationship to record this:

```
MATCH(s:Staff{name:'Ines Arana'}),(m:Module{module_id:'CMM510'})
CREATE (s)-[:teaches{position:'course_leader',academic_year:2018}]->(m)
```

This adds a 'teaches' relationship for 2018 between the nodes:

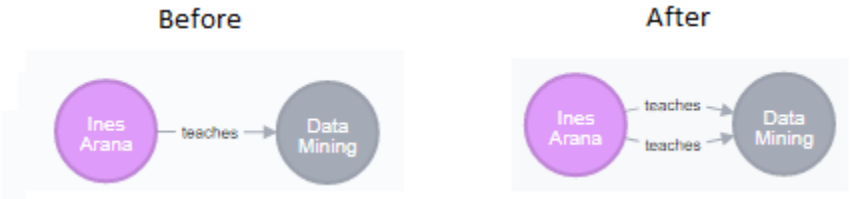

Fig. 4. Example result of adding data for a new academic year

Adding new relationships in this manner does not re-structure the graph as a whole. This means data for a new year can be added on an ad-hoc basis and queries written at one point in time will continue to work in the future. The exception is for the Cohort node which requires new nodes for every academic year. This is necessary since cohorts are unique to academic year.

3) The requested queries were successfully run. Minimising the node count at the expense of a higher relationship count may make future queries easier to write and understand.

Other design principles

A Year and semester entries are stored multiple times. This was a necessary compromise to allow all the nodes to contain unique data only. For reasons discussed in the design choice section, it is impossible to design a practical graph without some data replication.

B The above decision reduced the node count and makes for a simpler graph overall. It should be noted that a .grass stylesheet is also included with the final design to help improve readability.

C The requirement to keep hold of data from previous years complicated the overall design. However, the solution to only store date-related attributes in relationships means only new cohort nodes need to be added for a new academic year, along with new relationships.

D It is possible to update any combination of course/staff/modules in the given design (cohort is immutable since they are only defined once at the time of enrolment). By adding a new relation when a change occurs, the change is recorded and the previous state also remains available for record.

REFERENCES

[1] Robinson, I., Webber, J. and Eifrem, E. (2015). *Graph databases.*, 2nd ed. Sebastopol, CA: O'Reilly, pp.72-75.